NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THREE-DIMENSIONAL COMPUTER GRAPHICS VISUALIZATION OF TARGET DETECTION

by
Mehmet Gorgulu
Mustafa Yilmaz
December 1994

Thesis Advisor:	Richard Chris Olsen
Co-Advisor :	David R. Pratt

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE Three-Dimensional Computer Graphics Visualization Of Target Detection UNCLASSIFIED	5. FUNDING NUMBERS
6. AUTHOR(S) Mehmet Gorgülü - Mustafa Yilmaz	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT The purpose of this thesis is to visualize the sensor performance for a generic missile. We simulate the proceses performed by a missile using IR or TV sensors. Two generic scenes (background) were created, one for each generic sensor. The program simulates the scene from the point of view of a missile sensor. A graphical user interface was included for user input. These inputs provide the initial environmental conditions and the structural specifications of the sensor and the targets. Depending on these inputs, the sensor will show a detection and a lock-on range to the user. The detection range for the IR sensor was based on the intensity of the signal, above a specific threshold. For the TV system, target contrast was used. Atmospheric extinction was included. Several aspects of the SGI hardware and software capability were used to mimic physical problems and processes at considerable savings in computational effort. One was the use of the SGI Gouraud shading capability to establish the temperature distribution for IR targets; a second was use of the hardware (screen) projection to map from 3-D to 2-D. For further work, this program can be integrated to the EOTDA (Electro-optical Tactical Decision Aid) software. The graphics part of the program was written by using OpenGL graphics library and the user interface was implemented by using OSF/Motif. The main program was implemented in C++ on Silicon Graphics Reality Engines.

14. SUBJECT TERMS Infrared, TV, detection, lock-on, target	15. NUMBER OF PAGES 178
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

Three-Dimensional Computer Graphics Visualization of Target Detection

by
Mehmet Gorgulu
B.S., Turkish Naval Academy , 1988
Mustafa Yilmaz
B.S., Turkish Naval Academy, 1988

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN APPLIED PHYSICS

from the

NAVAL POSTGRADUATE SCHOOL
December 1994

Authors: [illegible]
Mehmet Gorgulu

[illegible]
Mustafa Yilmaz

Approved by: Richard C Olsen
Richard Chris Olsen, Thesis Advisor

[illegible]
David R. Pratt, Co-Advisor

[illegible]
for William R. Colson, Chairman, Department of Physics

ABSTRACT

The purpose of this thesis is to visualize the sensor performance for a generic missile. We simulate the processes performed by a missile using IR or TV sensors. Two generic scenes (background) were created, one for each generic sensor. The program simulates the scene from the point of view of a missile sensor. A graphical user interface was included for user input. These inputs provide the initial environmental conditions and the structural specifications of the sensor and the targets. Depending on these inputs, the sensor will show a detection and a lock-on range to the user. The detection range for the IR sensor was based on the intensity of the signal, above a specific threshold. For the TV system, target contrast was used. Atmospheric extinction was included. Several aspects of the SGI hardware and software capability were used to mimic physical problems and processes at considerable savings in computational effort. One was the use of the SGI Gouraud shading capability to establish the temperature distribution for IR targets; a second was use of the hardware(screen) projection to map from 3-D to 2-D. For further work, this program can be integrated to the EOTDA (Electro-optical Tactical Decision Aid) software. The graphics part of the program was written by using OpenGL graphics library and the user interface was implemented by using OSF/Motif. The main program was implemented in C++ on Silicon Graphics Reality Engines.

TABLE OF CONTENTS

ACKNOWLEDGMENT

We would like to thank Prof. Chris Olsen for his constructive and supportive guidance during our work. Also, we would like to thank Prof. Dave Pratt for his unlimited consultation. Finally, thanks to Prof. A. Cooper, for his consultation and for his generosity in providing resources during our research.

I. INTRODUCTION

A. A BRIEF DESCRIPTION OF OUR WORK

Visual effects have increased in importance in the last decade in both programming and interacting with computers. With the help of rapidly improving technology, we can put highly detailed graphics on our computer screens instead of only text characters. It has been shown that human perception is very high to visually presented data. As a result, computer graphics and visualization of data have been gaining an increasing importance in representing data visually. [Ref. 3]

Military platforms need sensors to perform their functions. These sensors are expensive to build and their performance is highly dependent on the environmental conditions and sensor characteristics. Any change in these characteristics changes the performance of sensors, and the ability of a sensor to perform its targeting function. Our work simulates this process and allows the study of the sensor performance at low cost, while varying environmental conditions.

For our work, we chose generic targets and backgrounds. We began with the physics problem of calculating the atmospheric absorption depending on the range, angle and wavelength and target projection area. We calculated the radiation for each node of the target and then found the power radiated by the target. By applying atmospheric extinction to this signal, we computed the signal level at the sensor. The temperature level at the sensor determined the temperature distribution on the target. The contrast value between the target and the background established the detection criteria for the target. There are different detection algorithms for the detection of the targets. These detection algorithms were examined and cross-box technique was applied in the program. Different computer graphics techniques were used in the program to be able to visualize the IR and TV scene and to be as close to reality as possible.

B. CHAPTER LAYOUT

Chapter II discusses the background and previous work in this area. It talks about the theory behind the modeling, different computer graphics and target visualization methods and the atmospheric extinction coefficient. The calculation of the atmospheric extinction coefficient and the use of Lowtran 6 including the reasons are described in this part.

Chapter III defines the problem and presents our assumptions. Chapter IV is an examination of the design decisions and the tools. It also covers the data structures for IR and TV sensors. Chapter V talks about the logical flow of the program and the components of the program, some of the problems and results. Chapter VI outlines the achievements in the program and makes suggestions on the future work of the program. Supplementary and detailed information about the program is presented in the Appendices.

II. PREVIOUS AND RELATED WORK

A. INTRODUCTION

A basic military problem in warfare is determining engagement ranges for weapon systems. Engagement is dependent on detection and recognition. The factors which affect the engagement range are environmental conditions, sensor characteristics, and the platforms which the sensor is on.

One tool which is used to prepare for engagement is the EOTDA (Electro-Optical Tactical Decision Aid) Software. This software has three sensor elements: Infra-red, TV and laser. There is a user interface to the program and data is provided to the program by this interface. This data includes the sensor type, environmental data and target type and target information. The program calculates the sensor lock-on range. The calculated data is displayed on the screen. The advantage of this program is it gives a good estimate of the detection and the lock-on range. The disadvantage is it was implemented on a PC. Hughes Aircraft Corporation implemented this program in Unix environment, but our brief Experience showed the program was not transportable.

We planned to work with the EOTDA software, and design a 3-D visualization of the EOTDA output. Unfortunately, the software is not allowed to be released to non-U.S. officers (allies). In line with our original intention, we decided to build some generic targets and backgrounds, and then work on detection ranges with generic IR and TV sensors. We modeled a tank, an aircraft, a ship and a building target with four possible backgrounds: soil, grass, concrete and sea.

We designed the program so that the user could choose the initial conditions for the program and then set the environment. The purpose of the user is to determine the detection and lock-on range for the IR and the TV system. In the program, the user was given the full control for changing the distance from the sensor to the target. As the distance changes, the calculations are done again depending on the changing distance. If the signal coming from either the background or the target exceeds the threshold level, the program gives detection.

The above implementation resulted in a computer program that can measure and visualize the performance of a generic IR or TV missile system depending on the input data. (The measure of performance for a missile system is detection distance.) The reliability of the result is completely dependent on the input data. The missile system gets the input and then tries to distinguish the target from the background and whenever there is detection, this is presented on the screen. The IR sensor part of the program we have, gives an idea of the temperature distribution on the target and the background. The user can see the temperature distribution on the target from different angles. The calculation method and the detection algorithm of the program rely on physical algorithms.

B. THEORY

1. Heat And Temperature

Heat can be defined as thermal energy in transition. It flows from one place to another as a result of temperature difference. The flow of heat changes the energy levels in the objects. Temperature is a property of the object and not a complete measure of the internal energy of the matter. Heat always flows from the object which is at the higher temperature to the object which is at the lower temperature. This is the principle governing the heat transfer between a target (heat source) and a sensor. [Ref. 14]

There are three models of heat transfer: conduction, convection and radiation. All heat transfer processes occur by one or more of these three modes. Infrared measurement is based on the radiative heat flow and is most closely related to the radiation mode of heat transfer. [Ref. 14]

Radiative heat transfer can take place across a vacuum and it occurs by electromagnetic emission and absorption. It occurs at the speed of light, and the energy transferred is proportional to the difference of the fourth power of the temperature. The bulk of the transport takes place in the infrared portion of the spectrum, from 0.75 to 100 μm. [Ref.14]

2. Infrared Radiation

The measurement of thermal infrared radiation is the basis for non-contact temperature measurement and thermography. The thermal infrared radiation leaving a surface is called exitance (radiosity).

Every object in nature emits wavelengths corresponding to its temperature. It is easy to get a temperature signature of an emitting object. The IR sensors are less susceptible to atmospheric conditions, and the frequency which the terrestial objects emit generally falls into IR frequency range. These are some of the fundamental reasons why the IR window is used.

All targets radiate energy in the infrared spectrum. The hotter the target, the more energy radiated. Very hot targets radiate in the visible spectrum. Infrared detectors can sense the infrared radiant energy and produce useful electrical signals proportional to the temperature of target surfaces. Every target surface above absolute zero (0 K or -273 ^{0}C) radiates energy in the infrared. When the targets are hot enough, they radiate or glow in the visible part of the spectrum.

As surfaces cool, not only do they emit less energy, but the wavelength distribution shifts to longer infrared wavelengths. Even though the human eye is incapable of sensing this energy, infrared sensors can sense these invisible longer wavelengths. They enable us to measure the self-emitted radiant energy from even very cold targets. [Ref. 14]

A blackbody radiator is an idealized source of radiation. Blackbody radiation has two important characteristic: Total energy, and the shape of the curve of intensity vs. wavelength (Location of peak.) Blackbody radiation is perfectly diffuse and radiates at all wavelengths. A blackbody source has an emissivity of 1.0 but the real sources in the nature have varying emissivities. In our program , we accepted our targets as blackbodies and made our calculations accordingly.

Every blackbody has a spectral emittance curve. Figure 1 shows a family of blackbody spectra, varying with temperature. These spectral curves were explained by Plank, at the turn of this century. [Ref 13] Equation 1 gives the power radiated per unit surface area per unit wavelength, as a function of λ.

$$M_\lambda(T) = \frac{2\pi c^2 h/\lambda^5}{e^{(hc/k\lambda T)} - 1} \quad \text{Watts} - \text{cm}^{-3}; \quad (1) \quad \text{[Ref. 7]}$$

In this equation h is Plank's constant, c is the speed of the light, k is the Boltzman constant, and λ is the wavelength at which the blackbody radiates energy. Equation 1 can be integrated to give the total radiation in a given wavelength range when integrated over all λ we obtain

$$W = \int \frac{2\pi c^2 h/\lambda^5 d\lambda}{e^{(hc/k\lambda T)} - 1} \quad = \quad \delta\varepsilon T^4 \quad (2) \quad \text{[Ref7]}$$

This is Stefan-Boltzman Law, obtained empirically by Stefan in 1879. The peak wavelength of the radiated energy is dependent of the temperature and is given by Wien's displacement law :

$$\lambda_m = b/T \qquad (3) \quad \text{[Ref. 5]}$$

In the above Equations :

W= Radiant Power emitted per unit area

ε = Emissivity

δ = Stephan – Boltzman constant

T= Absolute temperature of the target (K)

λ_m = Wavelength of maximum radiation

b = Wien's displacement constant = 2897 K

The sensor design we implemented responds to 8-12 μm bandpass. The proper approach would be to integrate Equation 1 over the λ range. Because the integration process slows the program down substantially, we assumed that the majority of the energy was 8-12 μm band, and simply used the Stefan-Boltzman Low to calculate the total radiance of the targets. Note that for terrestial temperature ranges, the radiation in any IR band varies linearly with temperature [Figure 1, and also Ref 13, page 121.]

The result of Equation 2 gives the radiation at zero range. The radiance we can use is the radiance normal to the surface of the target and this is found by dividing total radiant energy by π. So, $N=W/\pi$, N being the irradiance of the surface. Then the irradiance of the surface at a given range R from the object is:

$$P_R = \frac{WA_0}{\pi R^2} = \frac{NA_0}{R^2} \quad \text{W/m}^2 \qquad (4) \quad \text{[Ref. 5]}$$

In this equation, R is the distance between the target and the sensor, and A_0 is the projectional area of the target on the sensor. The energy at the sensor is calculated by multiplying the energy at the target by the atmospheric transmission factor. In our program, we did not assume that the effect of atmospheric extinction was simply inversely quadratic. We found the atmospheric extinction by using the Lowtran 6 code described below. This software gave us the atmospheric extinction coefficient .

The energy which is sensed by the sensor is also dependent on the projection area of the target on the sensor. The reason for target area dependency and how to calculate the projection area can be understood from Figure 2. The equation to calculate the projection area of the target is as in Equation 5.

$$A_T = l*h*\cos\theta*\cos\phi + w*h*\cos\theta*\sin\phi + l*w*\sin\theta \qquad (5)\ \text{[Ref. 5]}$$

In this equation A_T is the projected area of the target, θ is the elevation angle, ϕ is the azimuth angle, l is the length of the target, w is the width of the target, h is the height of the target.

3. Visual Wavelength Signals

In daylight reflected solar energy in the visible wavelength, can be used by TV sensors to detect the targets. Our implementation of TV detector depends on a contrast model. In this sense, contrast means the difference between the gray level of a target and the background. There is a sudden change of gray levels in transition from the environment to the target.

The energy coming from the target and the environment produces different illumination on the TV sensor screen. The energy striking the sensor plate induces different voltage levels for target and background. An electron beam scans this plate and creates different illumination values on the screen for each pixel as shown in Figure 3. Atmospheric factors affect this type of detection more than it does to other types of detections. We simulated this by applying fog in our program using an extinction model described below.

The software we used (OpenGL) provides us three types of fog functions. A fog function can be linear, quadratic or exponential. Since the atmospheric extinction occures exponentially, we chose the exponential fog function e^{-dR} . In this equation d is the fog density coefficient which corresponds to the atmospheric extinction coefficient in the IR part and R is the distance in eye coordinates from origin to the object.

TV tubes are a kind of electron device which converts an optical image into an electrical signal. These tubes are used to generate a train of electrical pulses which represent light intensities present in an optical image focused on the tube. These intensities correspond to the luminance or gray levels on the screen. Each picture element (pixel) on the screen has a luminance value.

These tubes use an electron beam to scan a photoconductive target which is the light sensor. A transparent conductive layer applied to the front side of the photoconductor serves as the signal (target) electrode. When a light pattern is focused on the photoconductor, its conductivity increases in the illuminated areas and the back side of the target charges to more positive values. The electron beam then reads the signal by depositing electrons on the positively charged areas thereby providing a capacitively coupled signal at the signal electrode. [Ref. 7]

The scene radiance is represented on the screen with different gray levels. The reflection of the scene on the screen will be represented by the pixels which have different brightness (luminosity) values. So there is going to be a pattern of contrast on the screen as shown in Figure 4 . The contrast among the pixels is related with their gray levels, i.e., with their luminosity. As a result of this there will be a distribution of contrast on the screen among the pixels. This contrast can be expressed in several ways. We define it as the difference of the maximum brightness and the minimum brightness. $C=GL_{max}-GL_{min}$ where GL_{max} is the maximum gray level, GL_{min} is the minimum gray level. [Ref. 6]. More information about the detection algorithm of TV sensors can be found in Chapter V.

4. Extinction Coefficient

The performance of military systems for imaging, target detection, tracking, target designation, and warning is strongly dependent on the transmission of the medium. The important atmospheric effects to be dealt with are refraction, absorption and scattering by the molecular constituents of the atmosphere. [Ref. 5]

The transmission characteristics of the medium in the measurement path between the target and the detector need to be considered in making noncontact measurements. No loss of energy is encountered when measuring through a vacuum. The atmosphere, however does not allow transmission in every band. As seen in the Figure

5, there are two spectral intervals which have high transmission. These are located in the 3-5 μm and 8-12 μm wavelength range. The IR sensors are built to scan in one of these atmospheric windows.

The effect of the atmosphere on the signal is represented by the extinction coefficient, which is the sum of the coefficients for total absorption and total scattering.

$$\mu_a = k_m + k_a \quad \text{and} \quad \mu_s = \sigma_m + \sigma_a \quad (6) \text{ [Ref. 7]}$$

where k_m molecular absorption coefficient

k_a aerosol coefficient

σ_m molecular scattering coefficient

σ_a aerosol scattering coefficient

In real world conditions, there are many factors that affect the transmission of atmosphere. For our program, we ignored scattering and the molecular absorption both for simplicity and lack of real data. To get the atmospheric coefficient we ran Lowtran in 8-12 μm. frequency band and got extinction coefficient values which are changing from 0.0703 to 1.166.

The TV sensor is affected by the same underlying rules. TV systems work in the visible part of the electro magnetic spectrum, as seen in Figure 5.

5. Lowtran

Lowtran is a FORTRAN Computer code designed to calculate atmospheric transmittance and radiance, averaged over a 20 cm^{-1} intervals in steps of 5 cm^{-1} in the spectral range of 350 to 40,000 cm^{-1}. The code uses single parameter band models for molecular absorbtion. [Ref. 8] Figure 6 illustrate the resolution of the code near 2200 cm^{-1} wave number. This is the 4,4 - 4,6 μm wavelength range with absorbtion due to N_2O and CO_2 [Ref. 12].

In our program, we used Lowtran to calculate the atmospheric extinction coefficient for different paths. The change of path in the atmosphere changes the extinction coefficient. For our program, the coefficients have been calculated for each 5^0 increments from 0^0 to 90^0 . The calculated data is kept in a different file and is read into the program interactively depending on the angle value coming from the top view angle slider. In Figure 7, the plot of the calculated atmospheric extinction coefficients are seen. These are the calculated values at 4 km range for each top view angle theta where the measured angle is the angle from zenith to horizon as seen in Figure 8.

6. The Detectors And The Detection Algorithms

The detector is a sensor which measures the energy that reaches to it. The energy which reaches the detector differs from the zero range value because of atmospheric extinction. The IR detector scans within a certain frequency range. The energy is collected by a lens and directed on the detector. An infrared interference filter is placed in front of the detector to limit the spectral region or band of the energy reaching the detector. The detector generates voltage which is proportional to the energy arriving from the target. This voltage is processed through some electronic circuits and then changed to target surface temperature. [Ref. 5]

Sensors scan the field of view. The scan techniques of sensor changes the structures of the sensors. There are three scanning techniques: Parallel, serial, serial-parallel. [Ref.7]

There are a variety of detection algorithms for IR sensors. The most known ones of these algorithms are :[Ref. 9]

- Contrast Box Algorithm, as shown in Figure 9 which depends on the contrast difference of the background and the target within the gate.
- Double gated filter algorithm is like a contrast box but uses a non-linear double gated contrast filter to localize the target.
- Spoke Filter depends on scanning the image of the target and it examines the locally complex gradient phase angles and gradient magnitudes.
- The Superslice algorithm employs multiple gray-shade thresholding and edge-matching to generate and segment possible target regions. In this algorithm, scanning is directional as shown in Figure 10.

The detection algorithm we used is similar to the contrast box model. The program reads the red value in the screen pixels, averages them and decides on the detection of the target.

While IR detectors work with temperature, TV detectors work with luminosity. Luminance is often called brightness, $L=dI/dA\cos\theta$. This is the luminous intensity per projected area normal to the line of observation. For TV systems, the radiation is at the visible part of the electromagnetic spectrum. Sources of light enlightens the targets and targets reflect the incoming light which is picked up by the sensors. [Ref. 7]

C. COMPUTER GRAPHICS

Computer Graphics started with the display of data on hardcopy plotters and cathode ray tube screens. Today, it has reached an amazing point with the help of sophisticated hardware and improved software and rendering techniques. It has grown to include the creation, storage and manipulation of models and images of objects. These models may come from diverse and expanding set of fields, and include physical, mathematical, engineering, architectural and natural phenomena. Computer graphics today is becoming largely interactive with the advent of virtual reality. [Ref. 2]

In the early 1980s, computer graphics was a small, specialized field because of the expensive hardware and software. Then personal computers with built-in raster graphics displays popularized computer bitmap graphics. In time, different techniques were developed and real objects were started to be drawn by polygons on computer graphics screens. The greater the number of the polygons, the more realistic was the picture of the object. Today, 80 million polygons per picture is accepted as the threshold for mimicing reality. [Ref. 3]

1. Visualizing a 3-D Target

As we mentioned above for the visualization of targets or objects, polygons are used as the simplest element (component) of the drawings. These polygons have vertices and the information about the polygons is stored in them. Generally, these polygons are in triangular shape. A user who wants to create an object must calculate the vertices of each polygon and then store these as data or must use special software to create the object. All of the polygons come together and they form the model. We can do lighting, shading and texturing on these models. [Ref. 3]

Every object drawn on the screen is approximated by polygons, and then these polygons can be rendered and with the help of lightning, texturing, and special graphics techniques, objects look real and 3-D. Each polygon in the object is called a facet. Facets have vertices. Our targets in the program are formed of polygons and the data about the target facets and vertex counts are presented in Table 1.

TARGET TYPE	VERTEX COUNT	FACET COUNT
T-62 TANK	1,384	960
F-16 AIRCRAFT	533	1,068
DESTROYER	628	590
HOUSE	84	28

Table 1. Target Geometric Information

The target models used in our work were developed by other students for previous projects either by Multigen modeling software or by hand-sketching. Multigen has a database which contains pre-drawn target files and different models. This database can be used to visualize different targets, but these models can not be used directly with OpenGL library functions. Therefore, we converted them to a standard form which can be understood by OpenGL library routines.

2. Texturing, Lighting, Shading and Fog

a. Texturing

Texturing is mapping a scanned image onto the surface of an object. This makes the image look more realistic. The techniques we apply texturing to a polygon are: [Ref. 3]

- The underlying color can be modulated with the texture image to get shaded textured models.
- The textured images can be mapped onto polygons, as a decal, obscuring the underlying polygon color. We used this method in our program. So, when light and texturing are on we do not see the underlying color. When the lights are off we do not see the color but the texture.
- A textured image can be blended with a single color.

By specifying the s&t texture mapping coordinates, [Ref. 10] we can cover the polygon with the desired image. In our program, we used texturing in the TV part to be able to get a real reflection of the world. In this part of the program, we used different textures to cover the environment and the objects. [Ref. 3]

b. Lighting

As in nature, in computer graphics everything reflects light and color. For the reflection of light, there must be a light source. In nature, the light source is the sun. In computer graphics the light source is created by the programmer. Lighting is required to give the objects a three dimensional and real effect. To use a shading model for 3-D objects, there must be a lighting model. Ambient, diffuse and specular illumination models are used in developed systems. In the real world, the perception of the eye depends on the distribution of photon energies that arrive and trigger the cone cells in the eye. The photons come from a light source, and these are either reflected or transmitted. This forms the basic idea of lighting in OpenGL. [Ref. 10]

c. Shading

Shading is a result of lighting. There are two types of shading: constant and Gouraud. In constant shading, a single color is computed for an entire polygon, based on the position of the light source and the normal vector of the polygon. Gouraud shading implies evaluating the illumination at each pixel level. The application of the shading to the objects gives them a realistic appearance. The shading model is used to calculate the intensity of the light at each pixel. [Ref. 3] In our work, we applied diffused and specular reflection rules to the targets in the TV part.

d. Applying Fog

Fog is a natural phenomenon and it affects the propagation of waves and the energy in nature. It works as an absorbent in the atmosphere. Even though, we do not have the absorption models in our program, we simulated this natural effect in our program. OpenGL provides a fog function. In our program, we determined the fog coefficient as a variable and, the user can control the density of the fog by using a slider in the TV (visible) mode. [Ref. 10]

III. PROBLEM

A. DEFINITION OF THE PROBLEM

The problem being approached here is two fold. We have a sensor detection problem, and a visualization task. There are two sensors with different detection algorithms.

As illustrated in Figure 11, for both the IR and TV sensor codes there is a target, a transmission medium, and a sensor. For our program, we implemented four targets: a tank, an aircraft, a ship and a building. This implementation can be broadened by creating more target files and making the necessary changes both in the interface and in the related programming routines.

The detection algorithms utilize emission and reflections for the IR and TV sensors, respectively. In each case, we model the effect of sensor field-of-view, sensitivity, and spectral range. Depending on their design properties and the ambient effects these sensors have different detection and lock-on ranges.

The energy radiated by the objects travels through the atmosphere toward the sensor. During this travel, due to the gaseous and particle material which are available in the atmosphere, the energy is attenuated and only a portion of it reaches to the sensor. Some portion of this energy is transmitted to the detector depending on the sensor parameters. This is the basic story of what is happening in our computer program.

Our work is a visualization of a real physical phenomenon. In summary, our problem can be defined as: the integration of a user interface design, the visualization of IR and TV targets under real physical conditions, and creating a tool which can be easily modified to measure the performance of different sensors.

B. ASSUMPTIONS

We made a number of assumptions to make the implementation easier in the short time period available. These assumptions do not divert the real output of the data from real calculations so much, but they do affect the precision. These assumptions can be taken as different aspects of the project and can be implemented to expand the project. The assumptions which are used in our work are as follows:

- The background of the target has a constant temperature. So, a fixed color value corresponding to this was applied to the background.
- The targets were assumed to be formed of nodes connecting facets to each other. Each of these nodes radiates energy.
- Each of the targets was assumed to be a blackbody.
- The environmental conditions for the targets were assumed to be mid-latitude summer conditions. Extinction coefficient calculations were made according to this assumption.
- The temperature range for the targets was assumed to be between 10-90 ^{0}C.
- The angle of approach of the missile was assumed to increase or decrease as multiples of 5 degrees within the range of 0-90 degrees.
- It is assumed that each target is a Lambertian surface, i.e. a surface which radiates in all directions.
- It is assumed that the energy which the sensor sees is proportional to the projection area of the target on the sensor.
- It is assumed that in the atmospheric conditions in which the energy travels, there are no aerosol.
- In the program, the delta t (time) interval is assumed to be very short for the events happening. There is no change in the temperature of the target during the flight of the sensor, and all of the extinction calculations are done depending on the initial temperature of the target. Making this kind of an assumption is not illogical, because the flight time of a missile is very short and the temperature change which may occur at the target during this period is negligible.
- It is assumed that the detection of an IR target occurs when the intensity of the signal (apparent temperature) exceeds the threshold level.
- It is assumed that detection of a TV target occurs when the contrast range in a small subset of pixels exceeds the threshold level.
- Lock on is accepted to occur if there are two separate detections on the same area on the screen.

IV. DESIGN

A. SELECTION OF WORKING ENVIRONMENT

1. Software

The program consists of three main components. The first part is the implementation of the user interface design, the second part is the sensor/detection physics involved, and the third part is the implementation of the (computer graphics) visualization.

For the user interface part of the program, we adopted OSF/Motif since this is as close to a graphics standard as is available. We used OSF/Motif for the user interface and to control the program input and output.

For the visualization aspect of the program, there were two tools we could choose IRIS Motif-GL or OpenGL. We used OpenGL because it is the most developed computer graphics library at hand. This software provides an interface to the graphics hardware and its implementation is easy. There are various commands and functions in the OpenGL library. It uses primitives and different algorithms for programming. It is more developed than the IRIS Graphics Library for developing a visual simulation program.

For the sensor physics element of the computer program, we used the C++ programming language. The main flow of the program was designed by using the C++ programming techniques. The rest of the code which uses special programming functions, was embedded into the main C++ code. The structure we used in the program was a simple multi-dimensional array structure to store and process the data.

2. Hardware

Computer graphics utilizes special hardware. In addition to rapid graphics processing, we utilized texturing for the TV sensor. Texturing requires fast computing and runs slowly on IRIS Indigo Workstations. Texturing is implemented in hardware on the IRIS Reality engines and runs much more quickly. So, our hardware working environment is the IRIS Reality engines. These workstation are very powerful and the data process rate of these machines are very high. The explanatory data about these machines are seen in Table 2.

IRIS POWER SERIES	4D/340 VGX
Number of the Microprocessors	4 33 Mhz IP7
FPU	MIPS R2010/R3010 Floating Point Chip
CPU	MIPS R2000A/R3000 Processor Chip
On-Board Serial Ports	2 per CPU board
Data Cache Size	64 KBytes
Instruction Cache Size	64 KBytes
Secondary Cache Size	256 KBytes
Main Memory	40 MBytes

Table 2. Specifications of IRIS Reality Engine

B. SENSOR DESIGN AND STRUCTURE

1. Infrared Sensor Design

The IR sensor is designed to mimic a threshold measurement technique in the 8-12 μm. atmospheric window. Objects are modeled using temperature distribution as described below. Only emitted radiation is modeled so, the sensor might best be considered a night - time model. The threshold level for sensors were chosen arbitrarily because of lack of real sensor characteristics. The unit for the threshold level is pixel intensity value within the range of 0 - 255.

The main idea for the IR part of the program depends on the representation of the temperature with colors. For this, there is a temperature scale on the screen, and the colors correspond to different temperature values of the pixels.

To implement the IR part of the program, we intended to write a program that simulates a generic IR sensor. We created some 3-D targets and applied color values to each vertex of these targets. This color on the target was interpolated from vertex to vertex and for each pixel on the screen between two points the temperature was extrapolated by the OpenGL software according to Gouraud shading as seen in Figure

12. This automatically provides an interpolated temperature distribution on the surfaces of the targets.

Since the radiated target temperature changes depending on the distance and angle between the target and the sensor, the color on each surface of the target changes as we increase or decrease the distance. The reason for this change is the atmospheric extinction. The program reads in the initial data file. Prior to drawing the object, the temperature value of each vertex is converted to a power value and extinction is applied to this power to calculate the power at sensor. The power at the sensor is converted back to temperature values. These temperature values are used as input parameters to figure out the color values from a look-up table. These color values are applied to the related vertices and the color of the whole target is changed.

The IR model depends on the relation between the temperature and the color values of the computer. The detection in the IR part of the program depends on this reading the color value of each pixel on the screen. The detection is determined by the hottest point or area on the screen. The hottest point on the screen is the point which has the highest averaged R value, i.e. the point which has the highest temperature.

The relation between the temperature and the color holds because in the color table presented in Chapter V, the R values of all colors were sorted in order. A color which has a small R value corresponds to a low temperature value.

2. Infrared Data Structure

In the program, we have four main arrays. One of them is used for storing the vertex data, one for storing the temperature of each vertex, one for storing the initial temperature of the vertex, and the last is used for storing the current (instantaneous) temperature of the vertices. Data is read into these arrays at the beginning of the program and then they are processed in these arrays. Vertex and initial temperature arrays are used once by the program at the beginning of the program. The current temperature and color arrays are used repeatedly during the program. Also the color values of the ground are stored in a global, one-dimensional array and updated with the target calculation procedures. The data transfer among these arrays is as seen in Figure 16.

3. TV Sensor Design

As we mentioned before, TV sensors generate electrical pulses which represent light intensities present in the optical image focused on the tube. The light intensities (luminance) are called gray level of the targets. The difference between the gray levels of a target is called contrast.

In nature, how we recognize objects is by detecting the contrast between the object and its background. So, we used the same idea in our TV detection algorithm. This idea is similar to IR detection, too. We used fog feature to emulate the atmospheric extinction because we were not using false color graphics implementation used for the IR sensor.

In the program, the whole screen of the computer was scanned and the contrast difference between pixels was checked. The area which gives the highest contrast difference was accepted to be the possible target.

C. VISUALIZATION DESIGN

The user interface in our program has two purposes.

- To control the input for the program.
- To test performance for different sensors whose parameters are different from each other.

There are a lot of parameters that control the infrared signature of the targets. These parameters are related with the temperature of the background, temperature of the target, atmospheric propagation conditions and the properties of the sensor. These parameters are in the user interface. The user can enter the data and measure the performance depending on these parameters. Inclusion of all of the programming routines that perform these calculations have not been completed in the program.

On the interface, the basic options for the user at the beginning are to choose the type of the target, the type of the sensor and background. The default values are a tank for the target and the IR for the sensor with the background of soil. After selecting these initial data, the display button displays the initial appearance of the target at the range of 4 km. We designed this user interface to give different options to the user and make the control of the program easier.

Another part of the user interface is on the display screen. There are three sliders on this screen. These sliders have been designed to control the input data of the program and the position of the program interactively and rapidly. One of these sliders controls the distance of the target to the sensor, the other slider controls the approach angle of the missile to the target. The last slider controls the viewing angle of the target temperature distribution. These sliders are not only used for controlling the appearance of target, but also as an input source to the program. We selected sliders as an input and control mechanism because, they are easy to use, fast and interactive.

V. THE PROGRAM

A. LOGICAL STRUCTURE

The problem of visualizing the output of EOTDA software consists of two parts. These sections are the IR part and the TV part. Therefore, we separated the problem into two general subsections at the beginning.

For the IR part of the program, the main hardship was first to be able to calculate the energy for the target model, second to represent of the temperature with the corresponding color value. To be able to solve these, we accepted a temperature range which can be expressed by the color Table 3.

COLOR NAME	R VALUE	G VALUE	B VALUE	TEMP. RANGE
Dark Blue	0	0	1	T<=10
Dirty Green	0.06	0.31	0	10<T<=15
Light Green	0.13	0.88	0	15<T<=20
Light Blue	0.19	1	1	20<T<=25
Brown	0.25	0.25	0	25<T<=30
Purplish	0.31	0	1	30<T<=35
Gray	0.38	0.38	0.38	35<t<=40
Dark Brown	0.44	0.58	0	40<T<=45
Cherry	0.5	0	0.5	45<T<=50
Fading Red	0.56	0.16	0.56	50<T<=55
Purple	0.63	0	1	55<T<=60
Dark Gold	0.69	0.82	0	60<T<=65
Dirty Red	0.75	0	0.4	65<T<=70
Dirty Pink	0.81	0.37	1	70<T<=75
Pinkish	0.88	0	1	75<T<=80
Gold	0.94	0.95	0	80<T<=85
Dark Red	1	0	0	85<T

Table 3. Temperature Color Values for the IR Part of the Program

The colors in this table were loaded into an array according to the temperature distribution on the target. Then the problem was to decide detection and lock-on precisely.

Since there is a one-to-one correspondence between the pixel colors on the screen and the temperature of the target, we can determine detection by using the R values of the pixels. In Table 3, it can be seen that the color values are sorted according to their increasing R values.

After working on the general theory of the problem, we made a design decision depending on the flow of the events in the main program. The main flow of the program for the TV and the IR part is as seen in Figure 13 and 14.

For the TV part of the program, what we see is close to the world as perceived by human eye. The detection is determined depending on the contrast difference between the target and the background. So, what we did was, we used a multi-dimensional array to store the contrast values of the screen, and then by finding the highest contrast box, we decided that there was a detection or a lock-on. The criteria for detection is, if the signal value of the screen pixel is above the chosen threshold level, then the program gives detection. If there is detection at the same screen location for a second time, then it is accepted as lock-on.

B. COMPONENTS OF THE PROGRAM

Table 3 contains a brief overview of the files in our program. The source code listings are contained in Appendix B.

FILE NAME	DESCRIPTION / EXPLANATION
Extinction_Coef.dat	The calculated Lowtran output for atmospheric extinction coefficient.
NPSimage.[C,.h]	Image handling routines.
*.rgb files	Scanned images of real objects which are used for texturing to cover the models.
display_list.h	Special OpenGL file which includes the drawing routines for the models.
draw_support.[C,.h]	The functions and their declarations which handle the drawing duty of the objects.
eotdamain.[C,.h]	Main flow of the program including the interface and the physical calculations of the program.
eotda_globals.h	This header file includes the global variables and the constants for the whole program.
house_list.C	Data for drawing the building target.
ship_list.C	Data for drawing the ship target.
plane_list.C	Data for drawing the aircraft target.
material_support.[C,.h]	Properties related with the lightning and the material properties of the targets.
texture_support.C	Functions related with texturing.

Table 4. The main files in the program

C. PROGRAM INPUT AND OUTPUT

The program takes files as input and outputs the visual color values on the screen and the numerical distance and angle values on the sliders. But as future work, a cursor connected to mouse movement can read pixel temperature values on the screen. So, there is no output file which is created by the computer. The input files which are read by the program load the arrays and these files are as listed in Table 5.

FILE NAME	DESCRIPTION / EXPLANATION
AircraftInitialFile.dat	Vertex number, 3-D coordinates and the initial temperatures of the facets of the aircraft model.
BuildingInitialFile.dat	Vertex number, 3-D coordinates and the initial temperatures of the facets of the building model.
ShipInitialFile.dat	Vertex number, 3-D coordinates and the initial temperatures of the facets of the ship model.
TankInitialFile.dat	Vertex number, 3-D coordinates and the initial temperatures of the facets of the tank model.

Table 5. Input Files and their Contents.

The files in the table above contains 3-D information about the coordinates of the targets and also information about the temperature of each node on the facets. The format of the input files is presented in Figure 15.

D. OPERATION

Our modeling and implementation of the program is summarized as follows:

The targets in the figures consist of facets. The more complex the target is, the more facets it has. The number of the facets and vertices belonging to each target has been presented in Table 1. Each facet on the target has vertices. These vertices are used to store information. The temperature value and the corresponding color values are stored in these vertices.

1. Infra-Red Sensor

- At the beginning of the program, we stored the temperature information of the targets in the vertices of the targets.
- We took each vertice on the target separately and calculated the energy radiated by this vertex at the stored temperature. This gave the energy of the vertex at zero range from the target.
- The energy value we found from this vertex travels in the atmosphere. During its travel, it attenuates. This attenuation is caused by atmospheric extinction. So, we multiplied this energy value with e^{-uR} . This multiplication gives varying values depending on the extinction coefficient and the distance between the sensor and the target. The result of this multiplication gives the energy value at the sensor.
- Sensors work by converting the energy value to temperature value. Hence, we converted this energy value to the corresponding temperature value. This corresponding temperature value is loaded into the current temperature array and the corresponding color is loaded into the color matrix. Then this color value is applied to this point for the next drawing cycle.
- The same cycle is applied to all of the vertices on the target successively. This procedure is the same for all the IR targets. After this procedure is done, the next step is to scan the screen for detection and lock-on.

At the beginning of the program, when we run the program, the program loads the arrays by reading data from the initial target files. The flow of data during the initialization phase of the program as shown in Figure 16.

In Figure 16, we can see what happens in the program when it is run. The first thing which is done in the program is to read the related target initialization data file. The data stored in these files are required for the initial visualization of the target. If the program is connected to another program like EOTDA software, and the output of the EOTDA software is provided in the same format, then would work as the initial target data.

The target initialization files contain data according to the sorted vertex numbers. Each of the targets in the program is formed of a lot of vertices, and data is stored in these vertices. In this file, data is in the order of vertex number, x-coordinate of the vertex, y-coordinate of the vertex, z-coordinate of the vertex, and the temperature value of the vertex. This data is in one line and the temperature in this line is the initial and the basic temperature of the target during the flight or cruise of the sensor. This temperature is the main temperature value for the rest of the extinction calculations.

After the vertex coordinates are read into the vertex array (the index of the array is accepted as the corresponding vertex number) the last value which is read in is the temperature value of that vertex. This is the temperature value of the vertex at zero range. These values which are in the initial temperature array are subjected to extinction calculations for each vertex, and then the newly found values are put into the current temperature array.

After the calculation of the new temperatures, the corresponding color values are stored into the color matrix and then the draw routine in the program is invoked. This provides an active color temperature calculation depending on the input parameters. The temperature related color values are applied to the vertices and appears on the screen representing the temperature of each point on the target.

The color values which are applied to the screen are the real values corresponding to the real estimated color values of the vertices i.e. the target. Because the color values have been sorted increasingly according to their R values, at low temperatures the corresponding color value on the screen is going to have a smaller R value, at high temperatures, the corresponding color value is going to have a high R value. This idea helps us when scanning the screen colors for detection.

The screen scan pattern was inspired from the contrast box method which was invented by Texas Instruments and the scanning method which is used in the program is as seen in Figure 17. [Ref. 9]

The scanning box on the screen covers an area of 200x200 pixels on the screen. This box moves on the screen from left to right like a scanner. At the end of each move, the red color values within the pixel area which is covered by the box are read, and the average value of the red values for the pixels in the box is found. These values are stored in an array and they are compared with each other in order to be able to find the hottest (the reddest) area on the screen.

The criteria for lock-on is the detection of the same point twice. Detection occurs when the biggest averaged red color value of a box exceeds the pre-accepted R value.

2. Television Sensor

TV sensors are designed to respond to a band of electromagnetic energy. The incoming radiation creates different level of illumination on the screen. This illumination is called the gray level, and the relation of the gray levels of the pixels gives the contrast.

Gray level for a TV system means the summation of the R, G and B values divided by three, i.e[(R+G+B)/3]. After finding the luminance values for each pixel in the scanning box on the screen, we took the value of the lowest and highest luminance in the box. By using the relation $C=GL_{max}-GL_{min}$, we found the C (contrast) value for each scanner box. Then this value is stored in the multi-dimensional array for comparison with the other box areas on the screen.

The values in each cell of the array, as shown in Figure 18, are the contrast values, x-coordinate and y-coordinate of the scanner box. We don't need the z-coordinate because we read the pixels of the screen. After storing all of the C (contrast) values for each box, these values are compared with each other and the largest contrast value is found. The x and y screen coordinates of the largest contrast box are accessible to us at any time, so we can detect at which point there is possible detection and we can plot this area on the screen.

Another condition for the detection to occur is that the C value must be above a threshold level. Having the same area as the highest contrast area for the second time results in lock-on.

E. SOME OBSERVATIONS

In Table 6 and Table 7, some observations of the program output are presented. In Table 6, the detection range for the tank target is shown. The distance values in the tables depend on the sensor characteristic and the environmental conditions. In this table the only chancing value is the approach angle of the sensor to the target. Since the approach angle changes the extinction coefficient, this angle affects the detection range of the sensor.

	APPROACH ANGLE (Degrees)			
SENSOR TYPE	0	10	45	90
IR	0.6 km	1.2 km	2.4 km	2.8 km
TV (No Fog)	4 km	4 km	4 km	4 km
TV (50% Fog)	2 km	1.8 km	2 km	2 km

Table 6. The Detection Ranges of IR and TV sensors for Tank target

In Table 6 only the approach angle of the sensor changes. In Table 7, detection ranges for different targets for 15^0 approach angle and soil background are presented.

SENSOR/TARGET	**TANK**	**AIRCRAFT**	**SHIP**	**BUILDING**
IR	1.6 km	3.2 km	1.2 km	2.8 km
TV (No Fog)	4 km	4 km	4 km	4 km
TV (50% Fog)	2.2 km	2 km	2 km	2 km

Table 7. The Detection Ranges of IR and TV Sensors for Different Targets

F. HOW DOES THE INTERFACE WORK

The user interface, as shown in Figure 19, provides input to the program and it consists of three main parts. The first part lets the user choose the target type and enter data about the target area, target speed and target course. From the target menu, we can choose from six targets. But we have implemented four of these targets in our program. The targets which are ready to be displayed are tank, ship, building and aircraft. The target area, direction and speed of the target areas are put on the interface for future use. In our program, we do not need these values.

The second part of the interface is the sensor part. In this part, we provide the sensor information to the program. The sensor type can have three different values. IR, TV and laser. We have implemented the first two of these sensors. For the IR sensor we have three different coloring scales. These scales are bi-level scale, gray-level scale and the cyclic scale. We designed the cyclic scale to present more accurate data on the screen. The user can decide the temperature of a point on the target by comparing the color values on the scale and on the target. Also in the sensor section of the interface, we have parameters which can change the detection possibility of the sensor. These parameters were put in the interface for the future development of the program.

The third part of the interface is the environment section. In this section, we provide information about the environmental conditions. The first information we provide is the background. We have a four background options in the program. These are soil, grass, cement and asphalt. The targets can be matched to one of these backgrounds.

The other parts of the environment adjusts the day of the year, hour of the day dependent on the sun's position, and the percentage of fog in the environment.

The display option displays the chosen target with the chosen sensor. Depending on the option which is chosen from the target and the sensor parts of the interface, the program switches to that output screen.

G. HOW DOES THE MAIN PART OF THE PROGRAM WORK

1. Infrared Scene

All of the implemented targets can be displayed in the IR screen. The IR screen shows the temperature distribution on the target. In the display screen, we have three sliders which let the user control the program and which also provide information to the program. Since the purpose of the program is to find the detection and the lock-on range depending on the sensor parameters, the user can play with the distance and the viewing angle of the target. Each distance and angle provides different input to the program because these parameters affect the atmospheric extinction coefficient for the program.

In Figure 20, the tank target is seen on the IR screen. On this screen, the exit button lets the user to get out of the program, the input button lets the user go back to the input interface and provide new data or chance information. The viewing distance slider lets the user control the distance between the sensor and the target.

We can visualize the target in different color scales. The cyclic scale in the display gives detailed information about the temperature distribution on the temperature. In Figures 21, 22 and 23, you can see the IR screen outputs of other targets.

2. Television Scene

The TV scene works as does in the IR screen. It is a reflection of the real world. The sliders available on the screen, function the same as it is in the IR screen. The TV screen outputs for the implemented targets are as shown in Figure 24, 25, 26, and Figure 27.

H. PROBLEMS IN THE PROGRAMMING

During our work, we experienced some compile errors which are machine dependent. In some machines we had z-buffer problem. By adding the line :

"XtSetArg(wargs[n],GLwNdepthSize,1);n++" in the main program, we got rid of this problem. This line initializes the z-buffer to a depth of 1 pixel.

We also observed that some of the scanned *.rgb files could not be read in some machines because of the difference in the graphics file. We saw that using display lists in OpenGL programming boosts the program. So, all texturing and hypertext fonts are encapsulated of display lists. We observed that as the size of the program expands, the program becomes more increasingly error prone.

To write error-free programs, we applied the following useful software engineering techniques:

- Modulation
- Encapsulation
- Top to bottom design
- Testing by control loops

- Well documentation
- Testing of each unit separately.

VI. SUMMARY AND CONCLUSIONS

The work which has been presented in this thesis is a visualization of IR and TV detection systems. This work was started with the intention of writing a visualization part of the EOTDA software. Due to the lack of real data, the project turned out to be a generic program for IR and TV sensors.

The program was written in C++ with OpenGL programming techniques and OpenGL graphics library functions. The interface part of the program was written in C++ and with OSF/Motif library function. The whole program was implemented under Unix operating system and the program can be transferred to other Unix systems if that system has the appropriate operating system and C++ and correct version of OpenGL libraries.

Computer programs similar to our project have been written by private companies, and their work took seven years to develop. This shows that there is a lot more to do on this computer program.

The program we wrote can be used for two purposes. The first usage might be the visualization of EOTDA software output, which would require that the output files of the EOTDA software be converted to the used format by the visualization program. The second usage might be a generic performance tester for different sensors and environment conditions.

The user interface provided at the beginning of the program provides input to the program. But some of the routines related with this input have not been developed in the program because of time constraints. So, these routines can be developed or improved and the whole program can be used as a performance tester of sensors which have different construction parameters.

The output file of the EOTDA file contains the facet number of each facet and the corresponding temperature to this facet. But the input file of our visualization program includes the vertex number, x, y, and z coordinates, and temperature of this facet. Since we were unable to obtain EOTDA data at the beginning of our project, we created our input file format and data for our own program.

The first thing that can be done on the program is that the program can be integrated to EOTDA software. This requires a small change in the output files of the EOTDA software.

LIST OF REFERENCES

1. Hughes Company, Electro-optical Tactical Decision Aid (EOTDA) User's Manual Version 3.0, 1983

2. Donald Hearn, Pauline Baker, Computer Graphics ,Mc Graw-Hill, 1990

3. Foley, Van Dam, Hughes, Computer Graphics, Addison & Wesley, 1994

4. Donald Mc Minds, Mastering OSF/MOTIF Widgets, Addison & Wesley, 1993

5. DCS Corporation, Infrared Imaging Systems Analysis, DCS, 1988

6. John C. Russ,The Image Processing Handbook, Osborne, 1992

7. RCA, Electro-optics Handbook, 1974

8. AFGL-TR-88-0177, U.S. Air Force Geophysics Lab, Lowtran 6 Code Manual, 1990

9. Texas Instruments, Infrared Detection and Sensors, IEEE, 1991

10. Silicon Graphics, OpenGL Programming Manual, San Jose, 1994

11. James A. Ratches, Static Performance Model for Thermal Imaging Systems, 1976

12. Alexander Berk, Lawrence Berstein, and David C. Robertson, Modtran: A moderate resolution model for Lowtran 7, USAF Technical Report GL-TR-89-0122, 1989.

13. Richtmyer & Cooper, Introduction to Modern Physics, Mc Graw-Hill, 1969

14. Arya, Fundamentals of Atomic Physics, Allyn & Bacon Inc., 1971

15. Silicon Graphics Inc., Iris Universe Magazine Summer 92

16. Elachi, Charles, Introduction to the physics and the techniques of remote sensing, John Wiley & Sona, New York, 1987

APPENDIX A. FIGURES AND SCREEN OUTPUTS

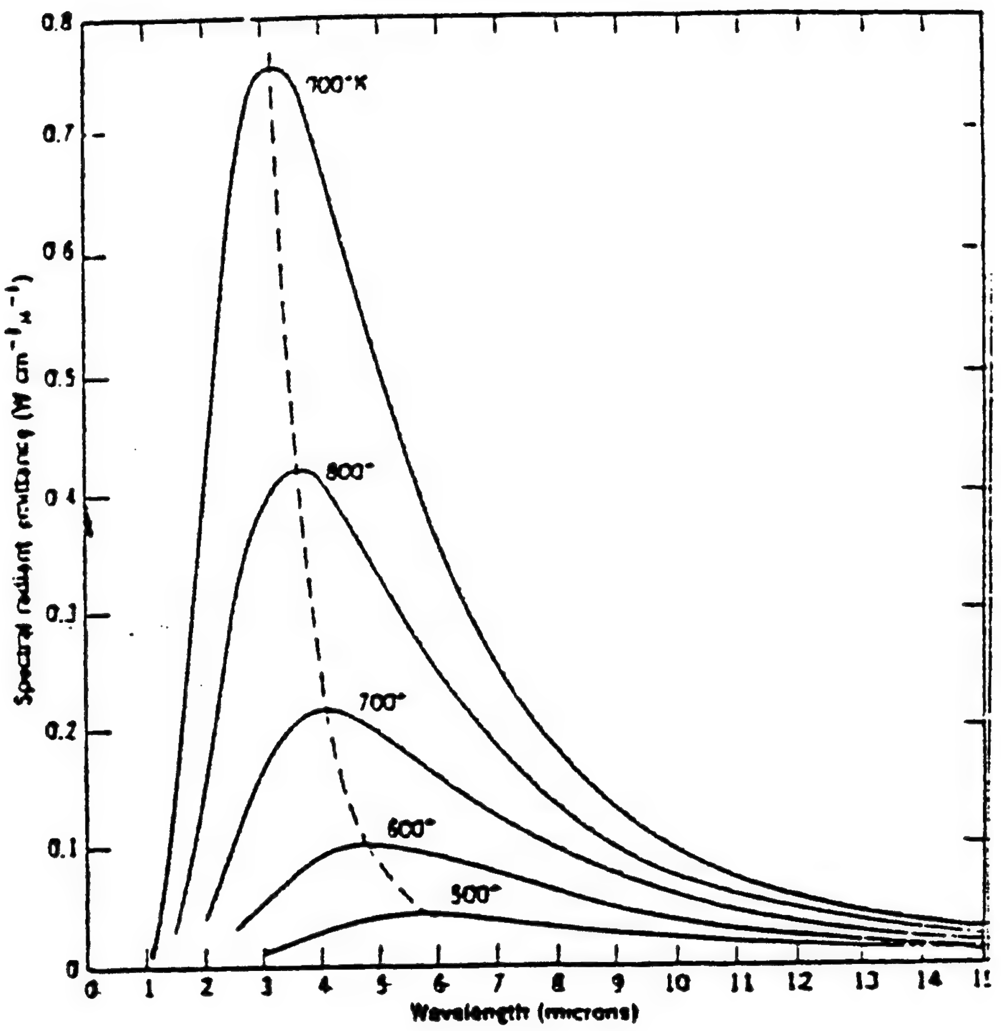


Figure 1. The change of the blackbody curves with temperature "From Ref[7]"

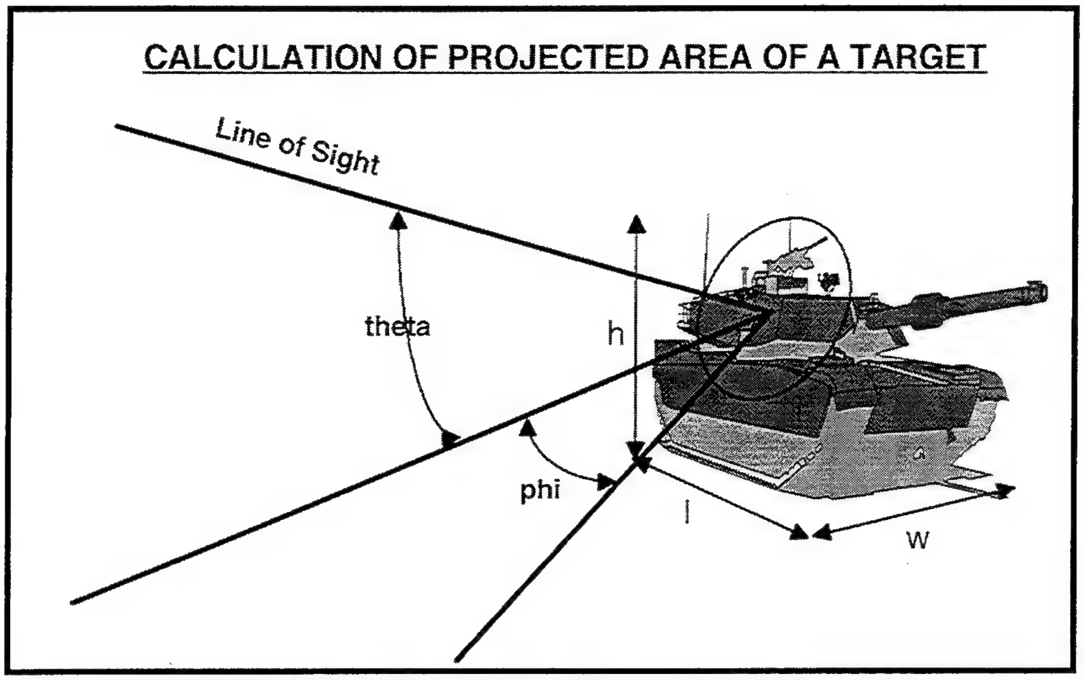


Figure 2. The calculation of the projection area of a target

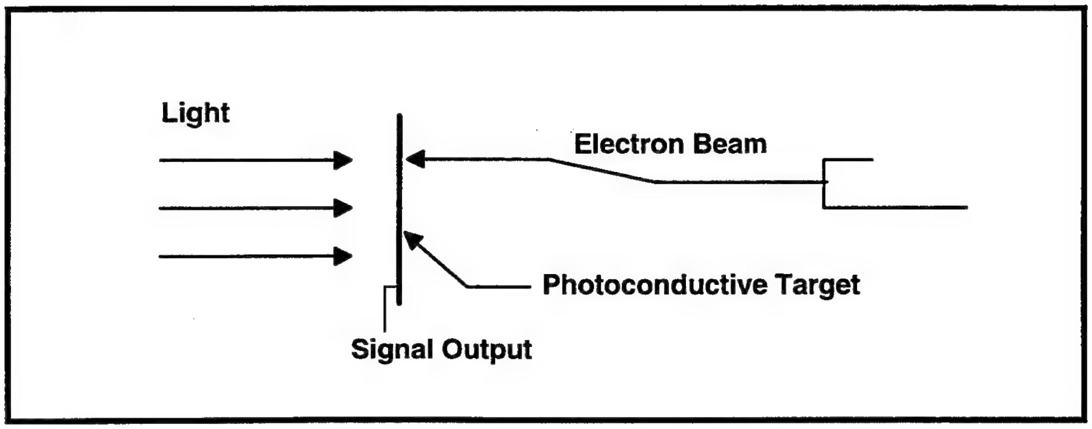


Figure 3. Schematical appearance of a TV camera tube "From Ref. 7"

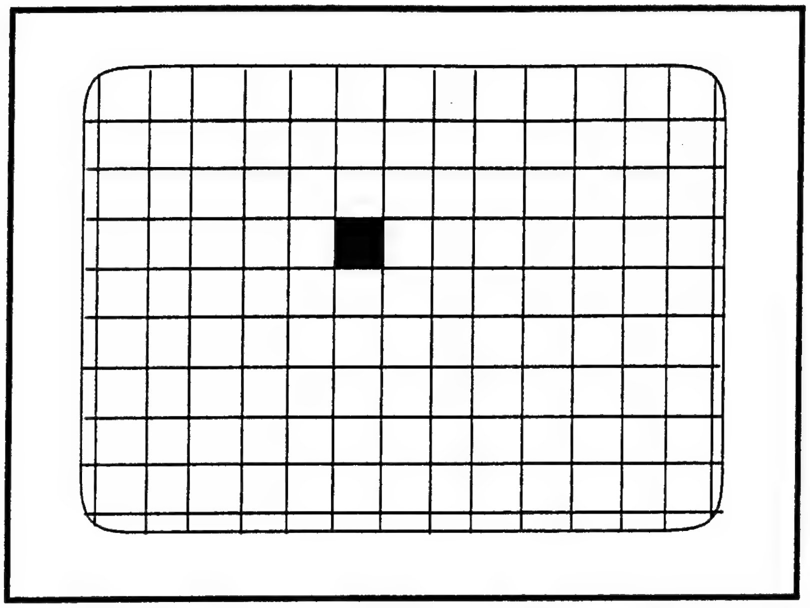


Figure 4. The Representation of Contrast Values on Computer Screen

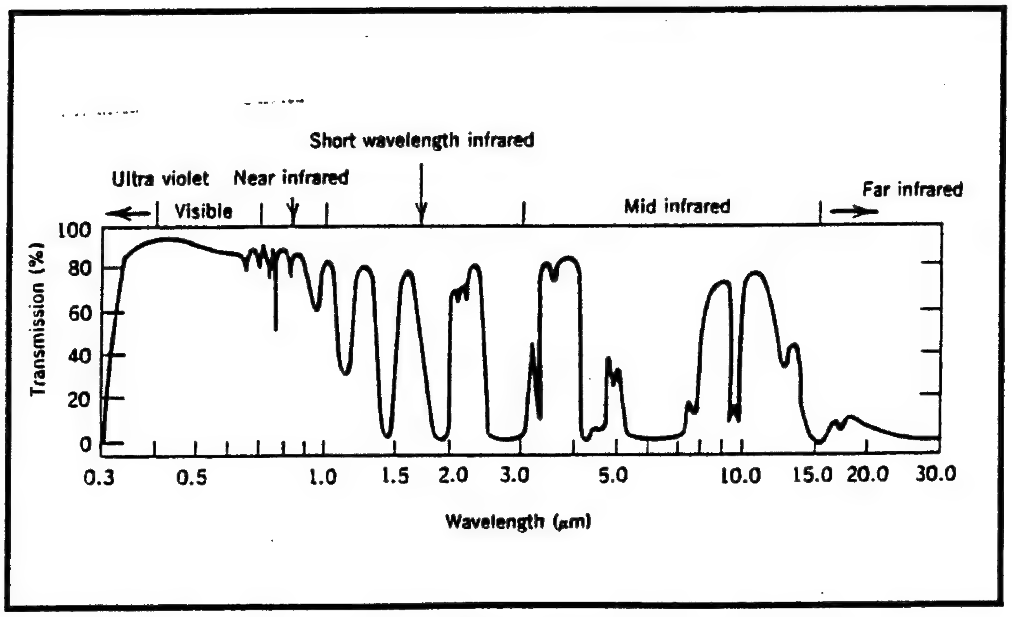


Figure 5. Atmospheric Transmission Windows "From Ref 16"

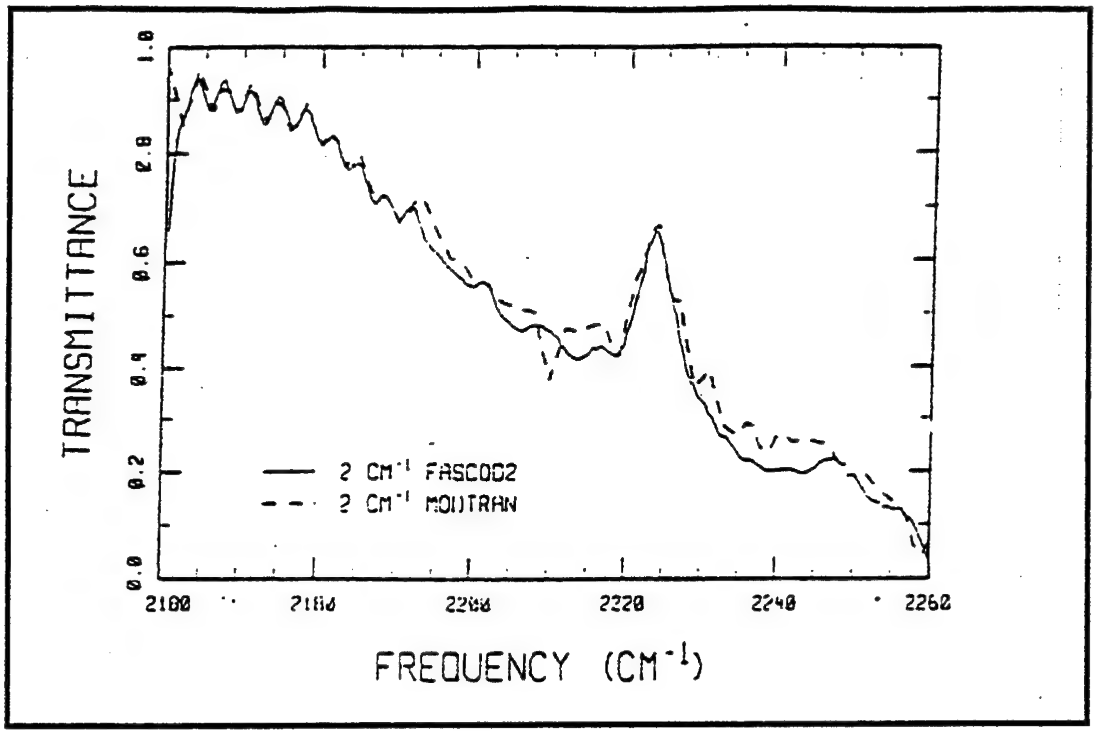


Figure 6 . Transmittance of the Athmosphere "From Ref. 12"

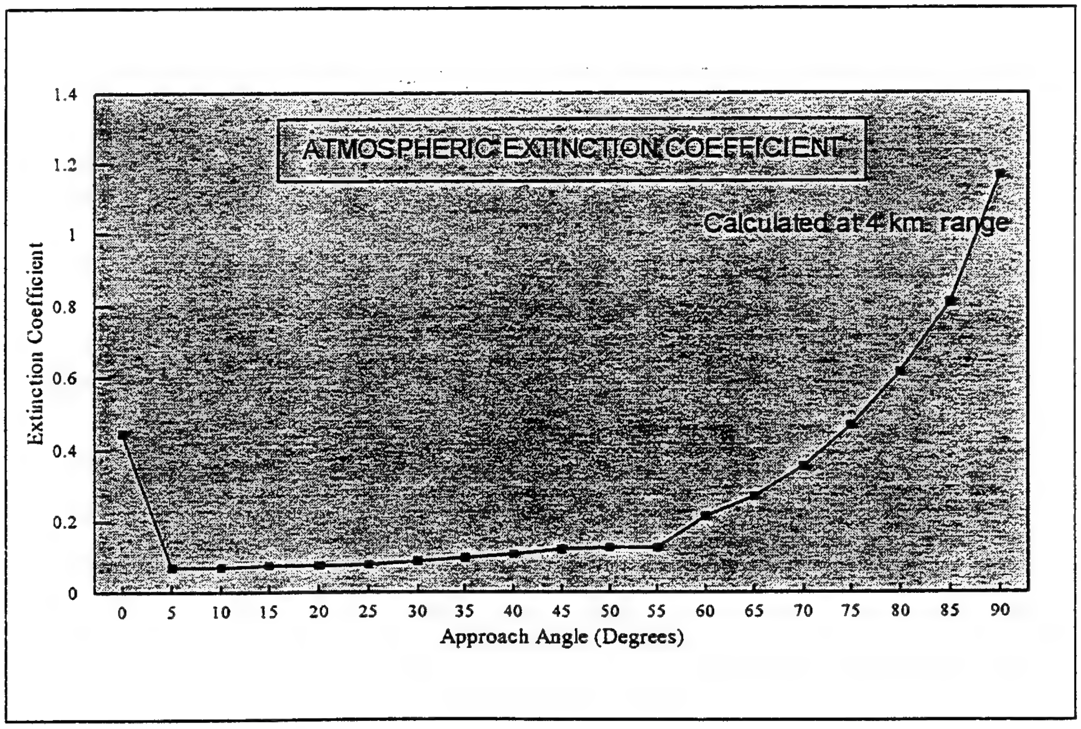


Figure 7. The plot of the atmospheric extinction at the range of 4 km. for 8-12 IR window

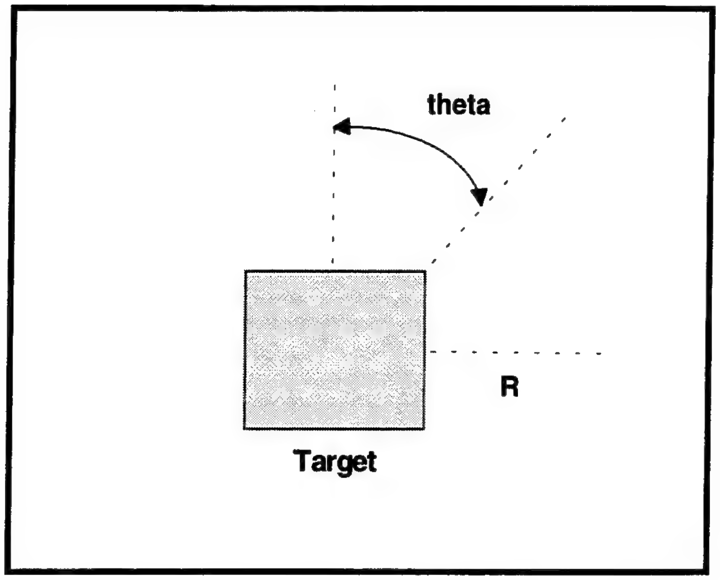


Figure 8. The measurement of theta in Lowtran Code

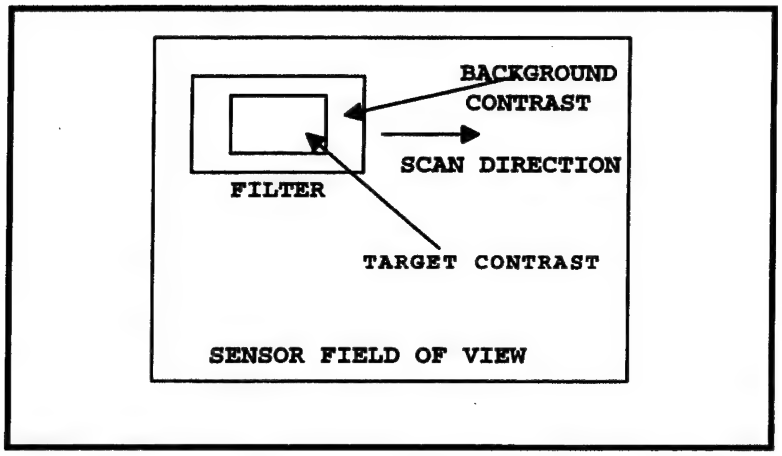


Figure 9. The Schematical Representation of the Contrast Box Algorithm

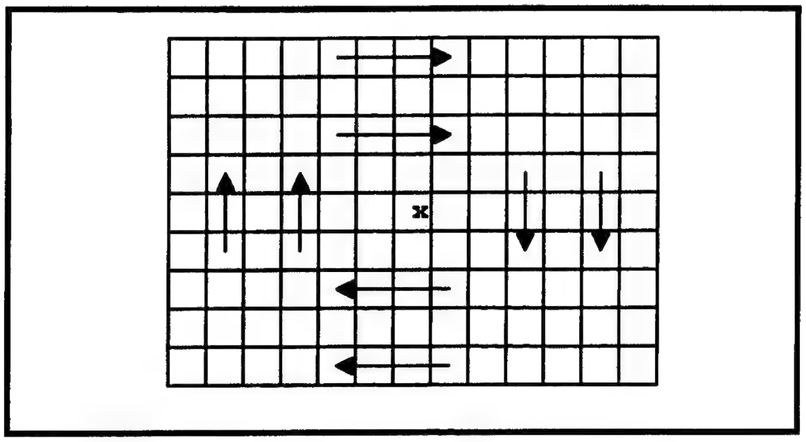


Figure 10. The Super-Slice Algorithm

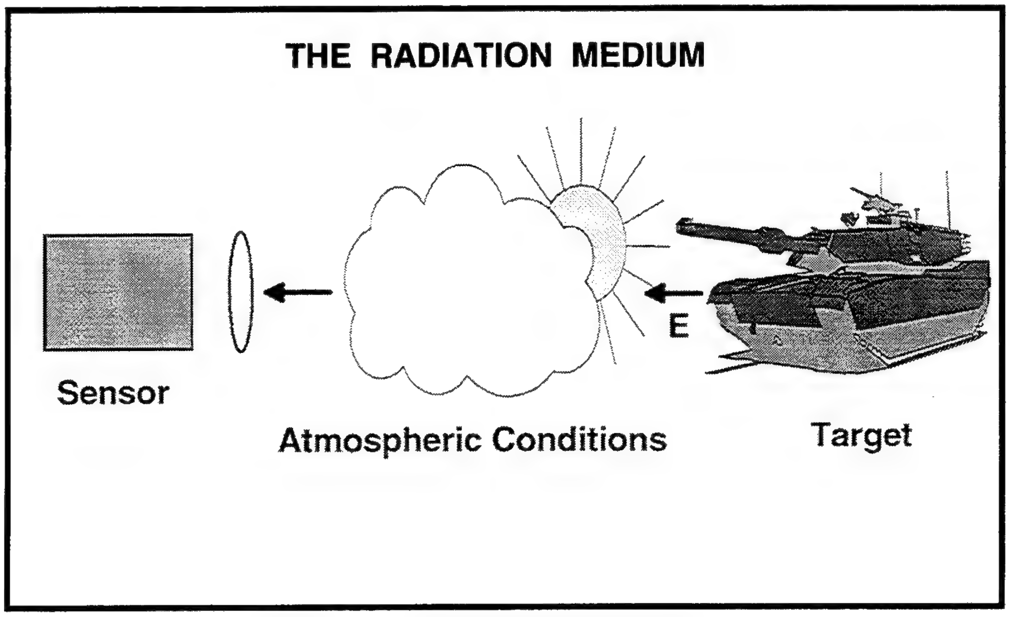


Figure 11. The Radiation Medium

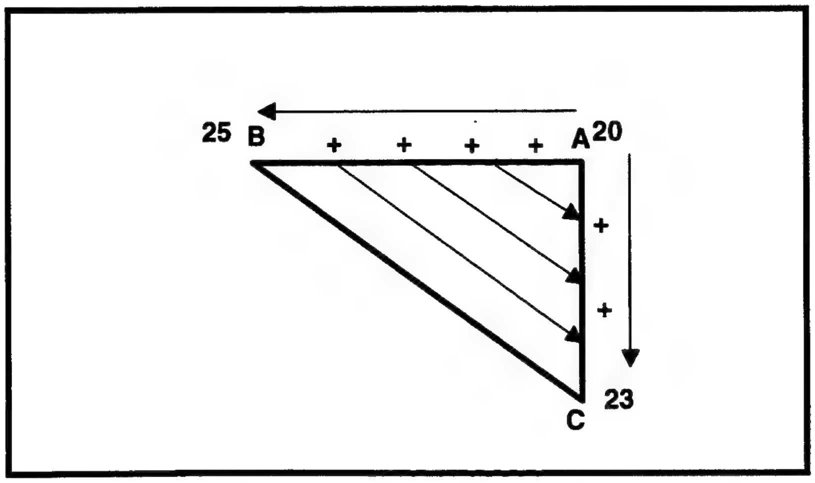


Figure 12. Automatic application of the Gouraud Shading by hardware

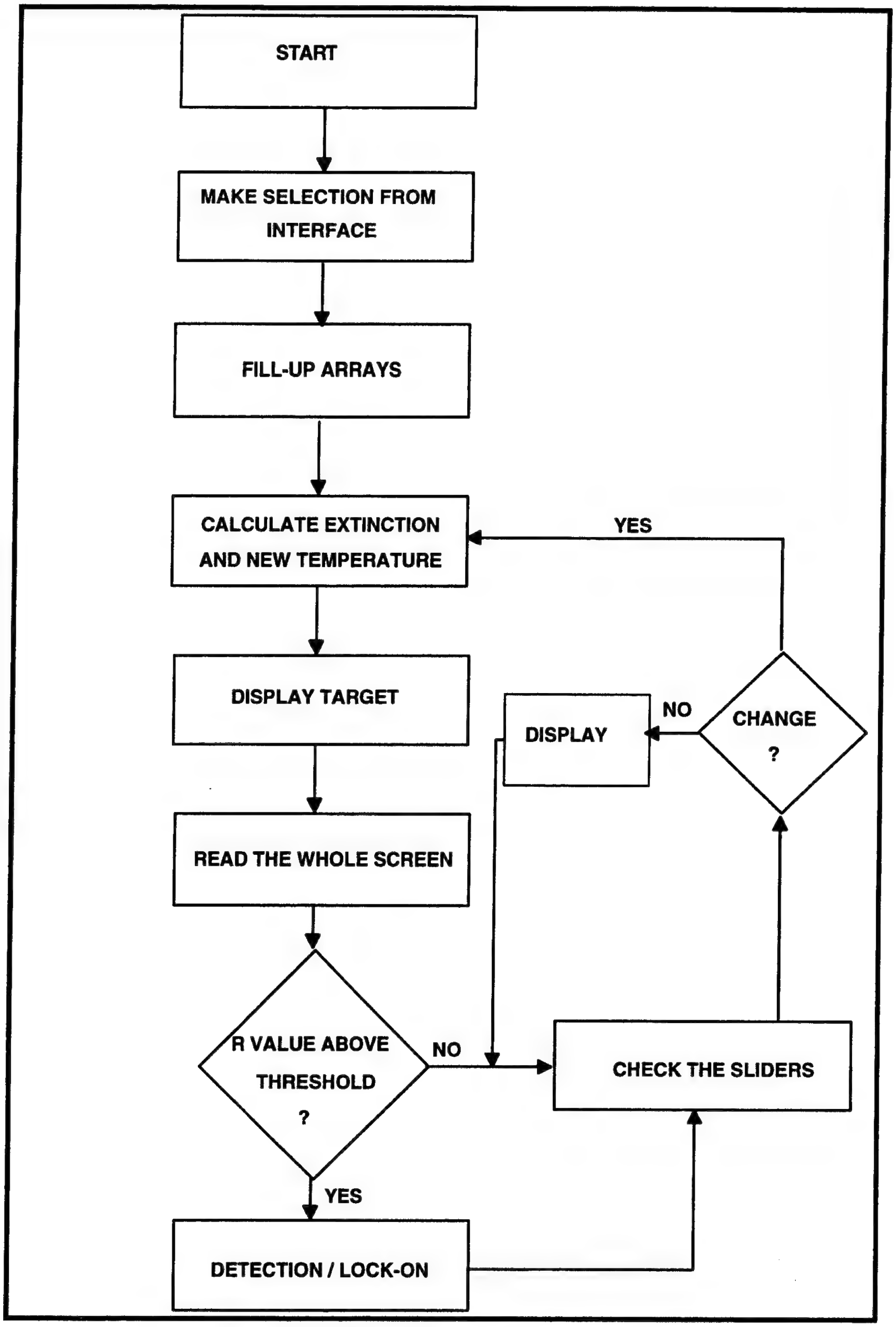


Figure 13. Infrared Sensor Logic Flow Chart

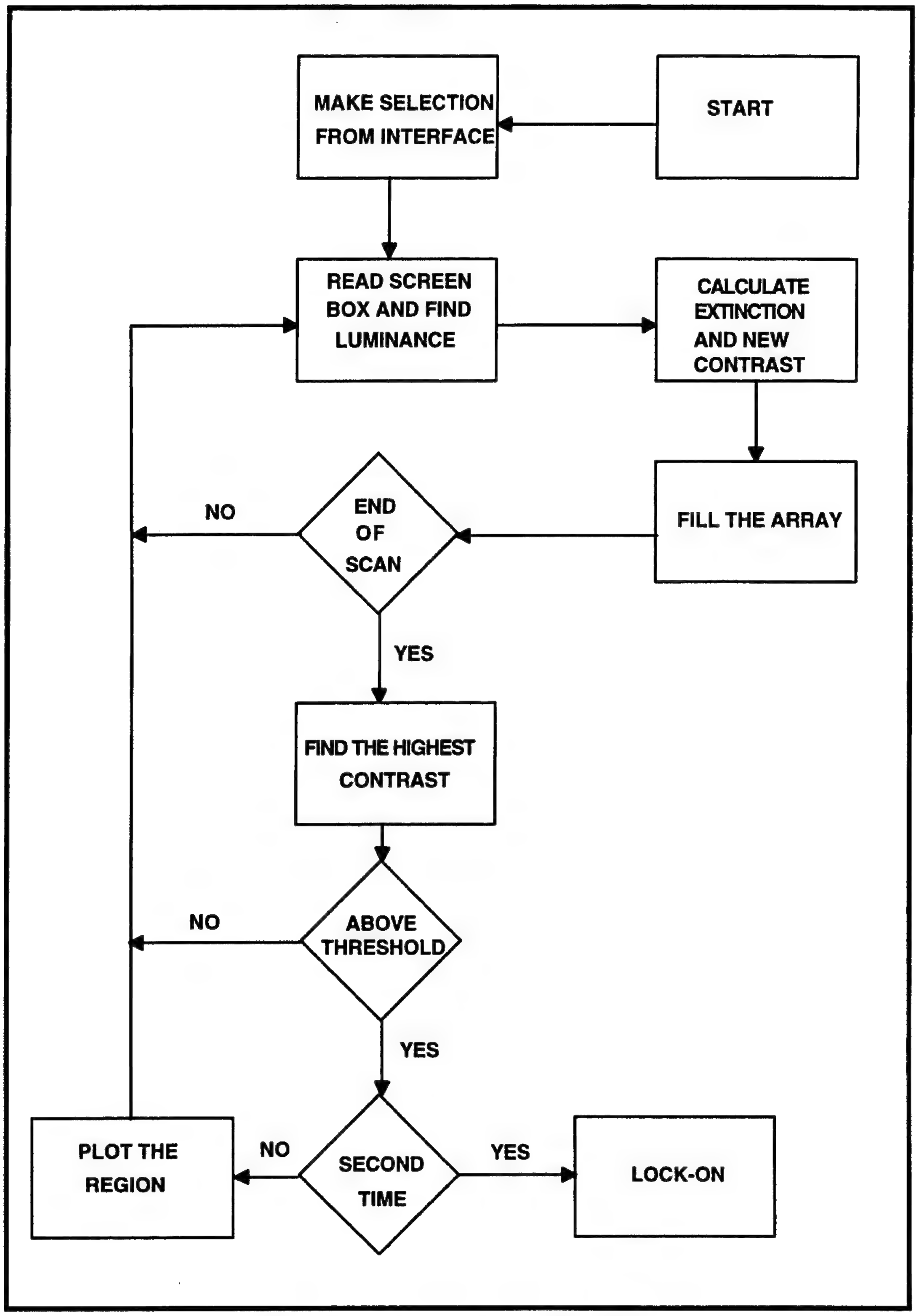

START
MAKE SELECTION FROM INTERFACE
READ SCREEN BOX AND FIND LUMINANCE
CALCULATE EXTINCTION AND NEW CONTRAST
FILL THE ARRAY
END OF SCAN
NO
YES
FIND THE HIGHEST CONTRAST
ABOVE THRESHOLD
NO
YES
SECOND TIME
NO
YES
PLOT THE REGION
LOCK-ON

Figure 14. TV sensor Logic Flow Diagram

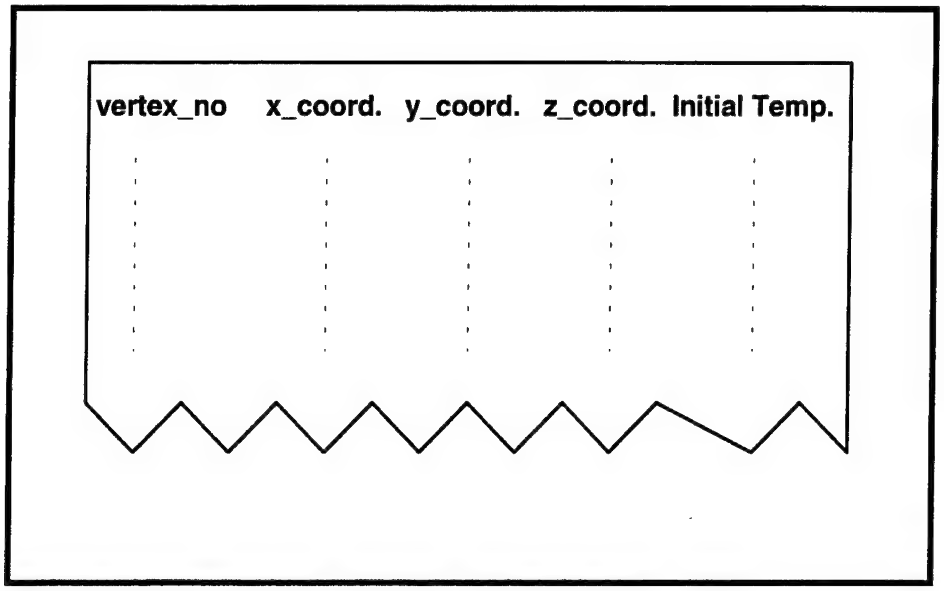


Figure 15. Input File Format

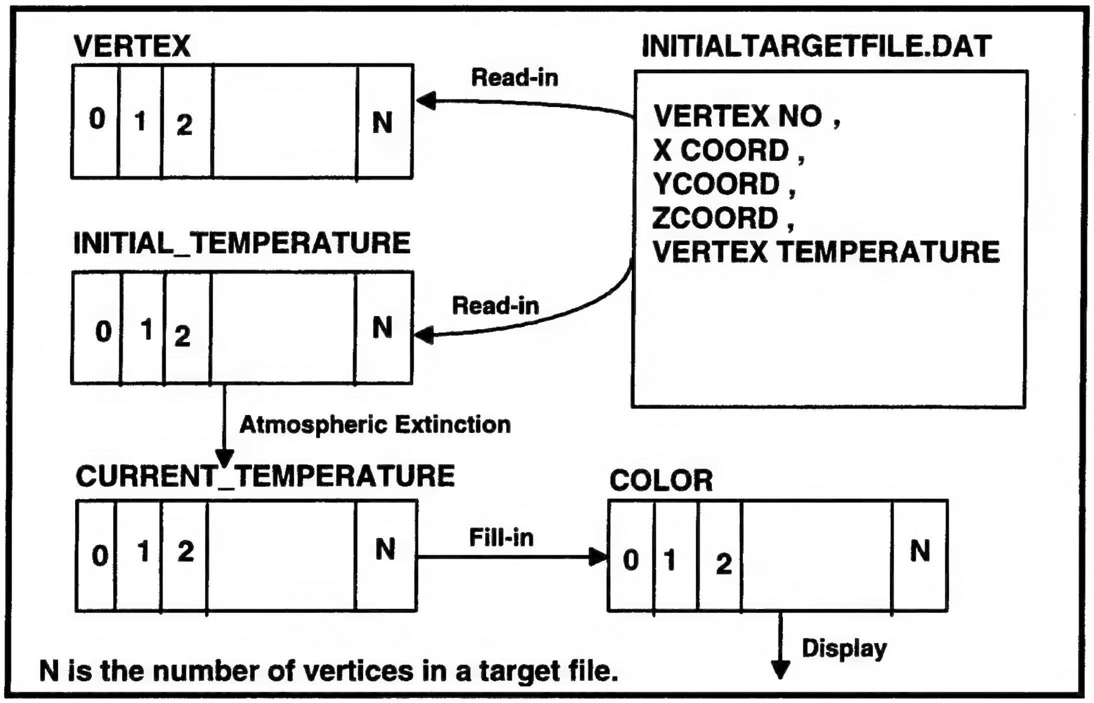


Figure 16. Process of Data in the Program.

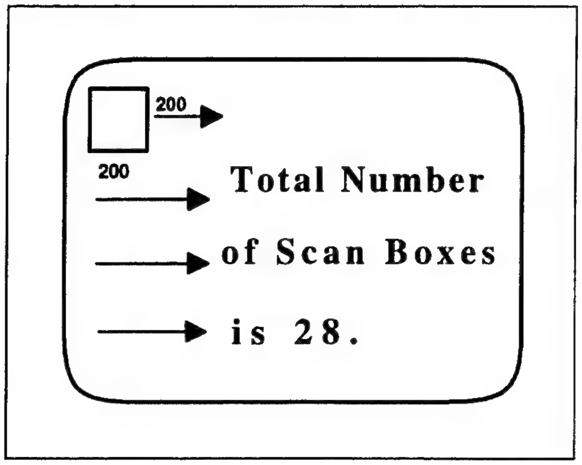


Figure 17. Scanning Box on the Screen

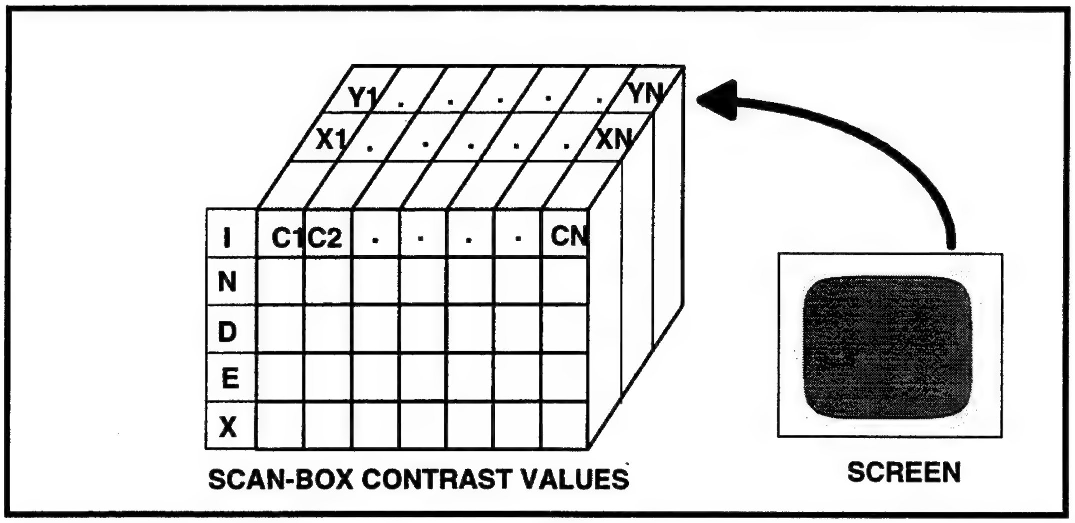
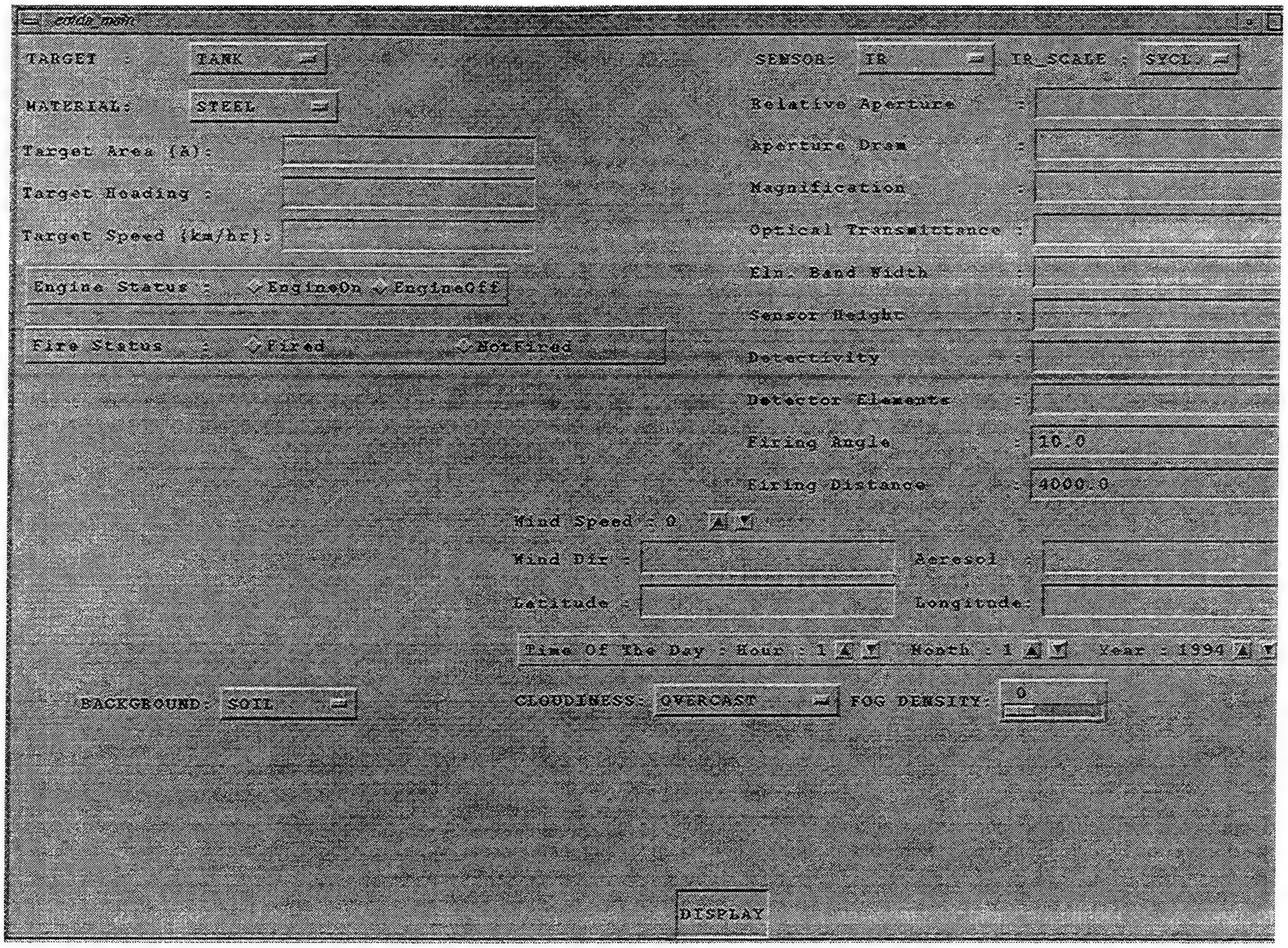


Figure 18. Storage of Contrast Values from the Screen.



```
TARGET  :  TANK                              SENSOR:  IR          IR_SCALE : SYCL
MATERIAL:  STEEL                             Relative Aperture      :
Target Area (A):                             Aperture Diam          :
Target Heading :                             Magnification          :
Target Speed {km/hr}:                        Optical Transmittance  :
Engine Status  :  ◇EngineOn ◇EngineOff       Eln. Band Width        :
                                             Sensor Height          :
Fire Status    :  ◇Fired        ◇NotFired    Detectivity            :
                                             Detector Elements      :
                                             Firing Angle           : 10.0
                                             Firing Distance        : 4000.0
                        Wind Speed : 0
                        Wind Dir :                        Aeresol   :
                        Latitude :                        Longitude :
                        Time Of The Day : Hour : 1   Month : 1   Year : 1994
  BACKGROUND: SOIL      CLOUDINESS: OVERCAST     FOG DENSITY: 0
```

Figure 19. The Screen Output Of The User Interface

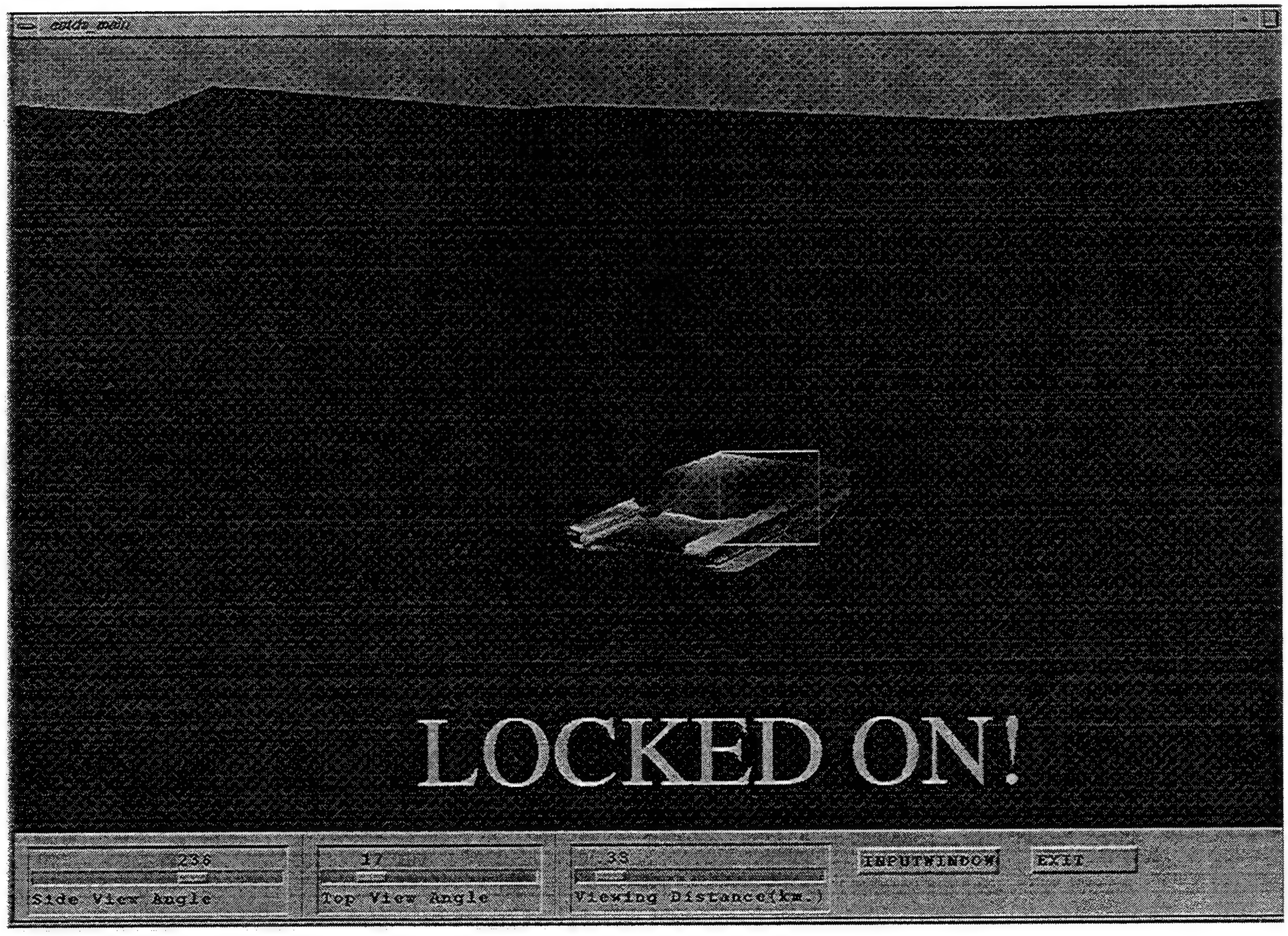
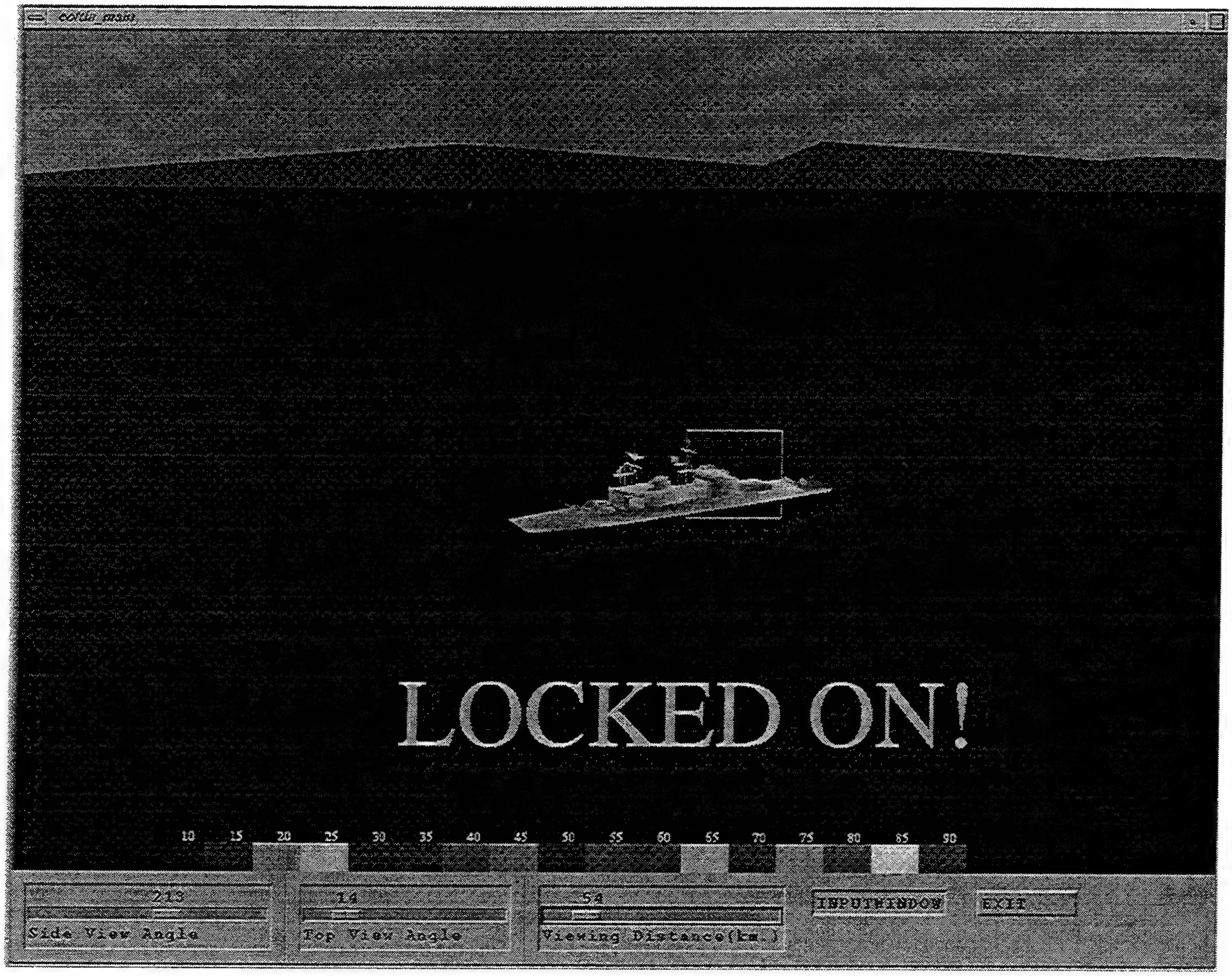


Figure 20. The IR Screen Output Of The Tank Target.



213
Side View Angle
14
Top View Angle
54
Viewing Distance(km.)

Figure 21. The IR Screen Output Of The Ship Target.

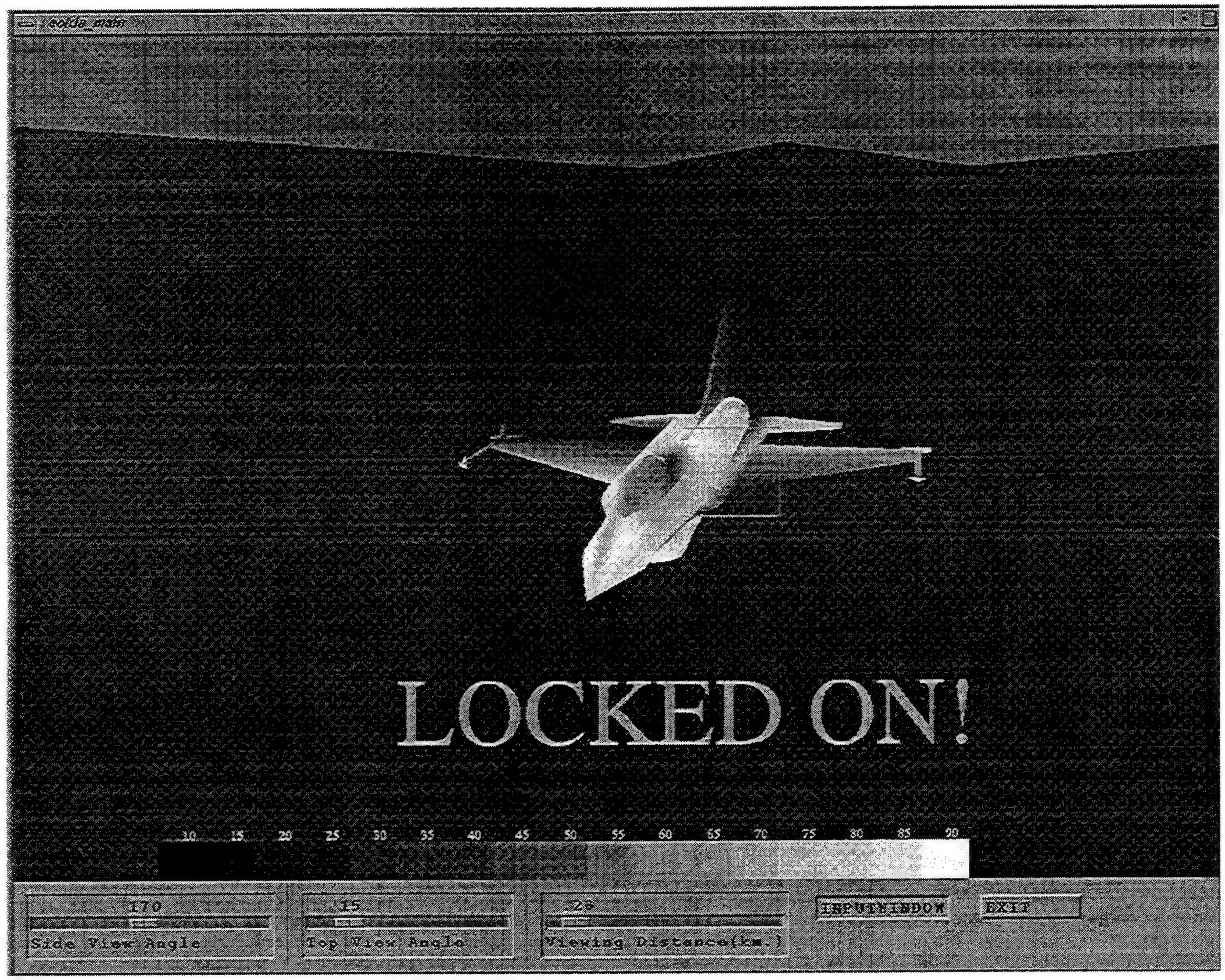


Figure 22. The IR Screen Output Of The Aircraft Target.

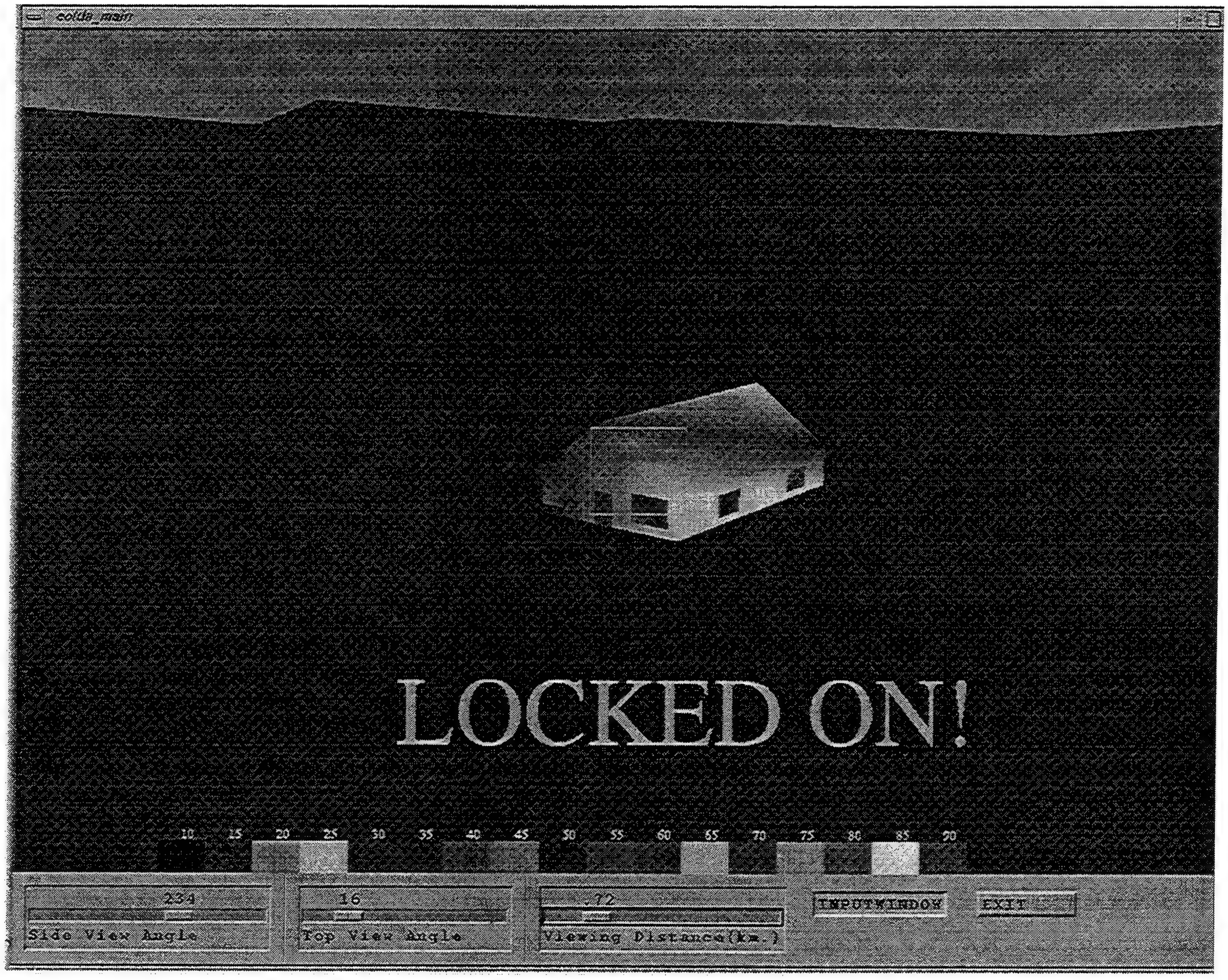


Figure 23. The IR Screen Output Of The Building Target.

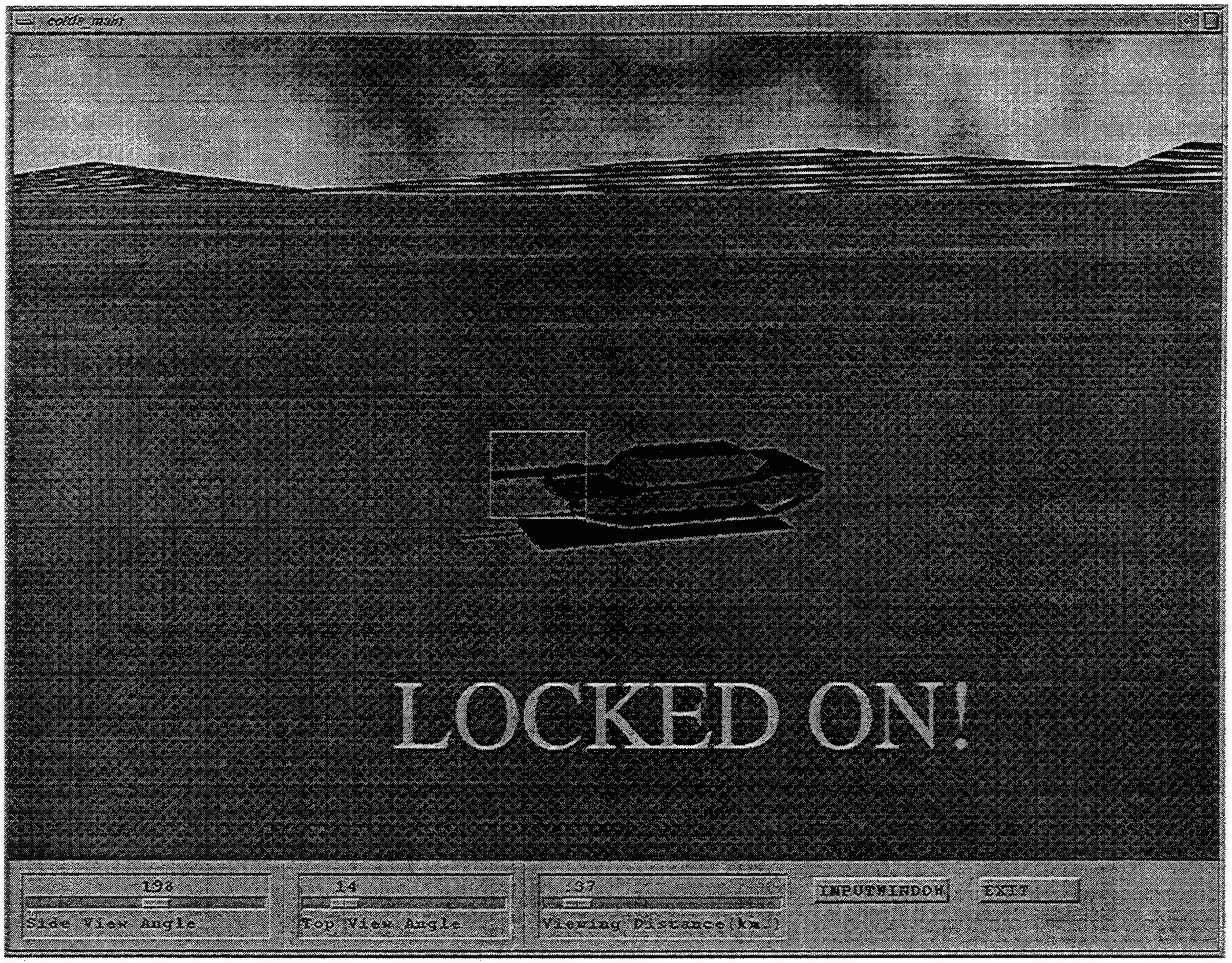


Figure 24. The TV Screen Output Of The Tank Target.

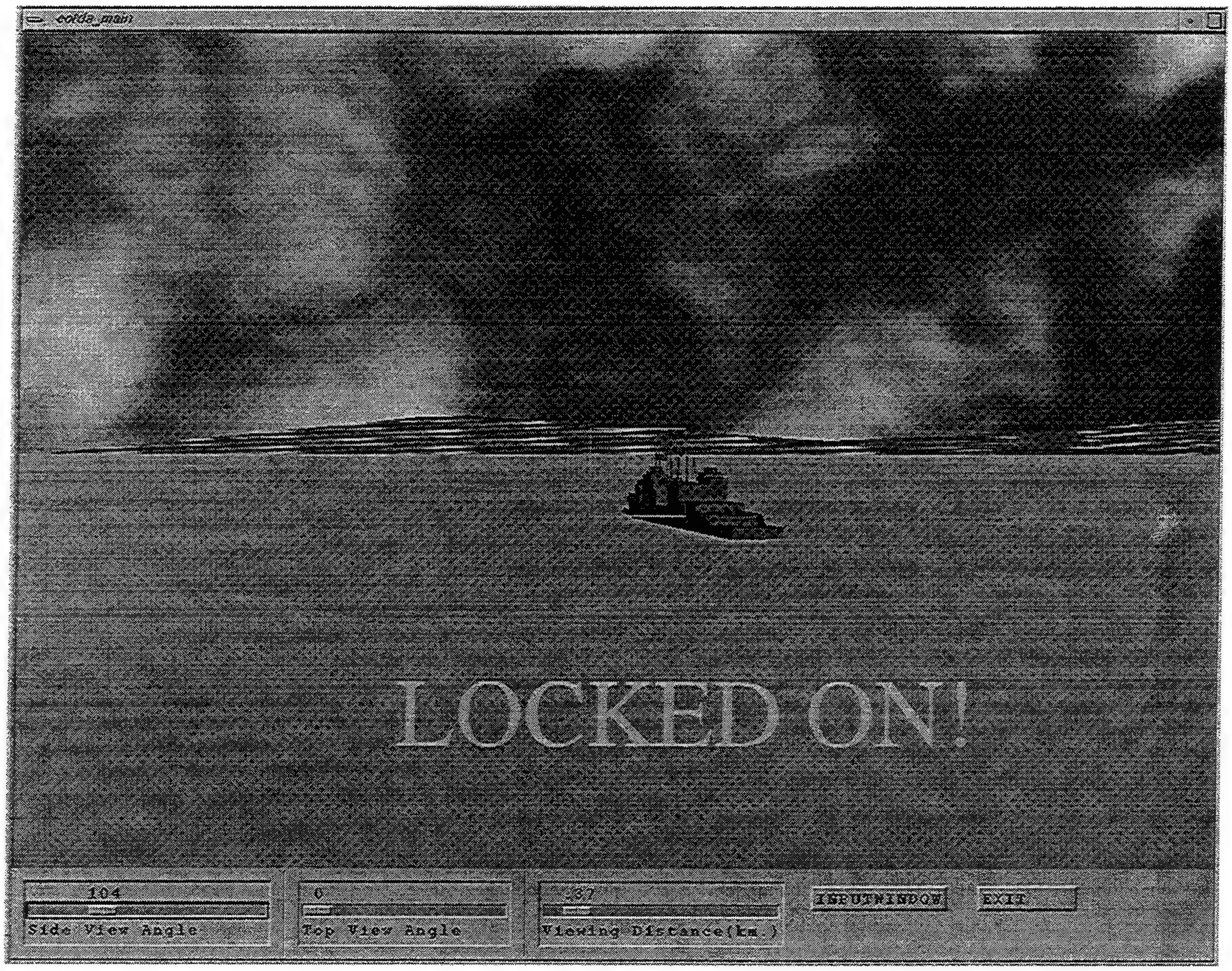


Figure 25. The TV Screen Output Of The Ship Target.

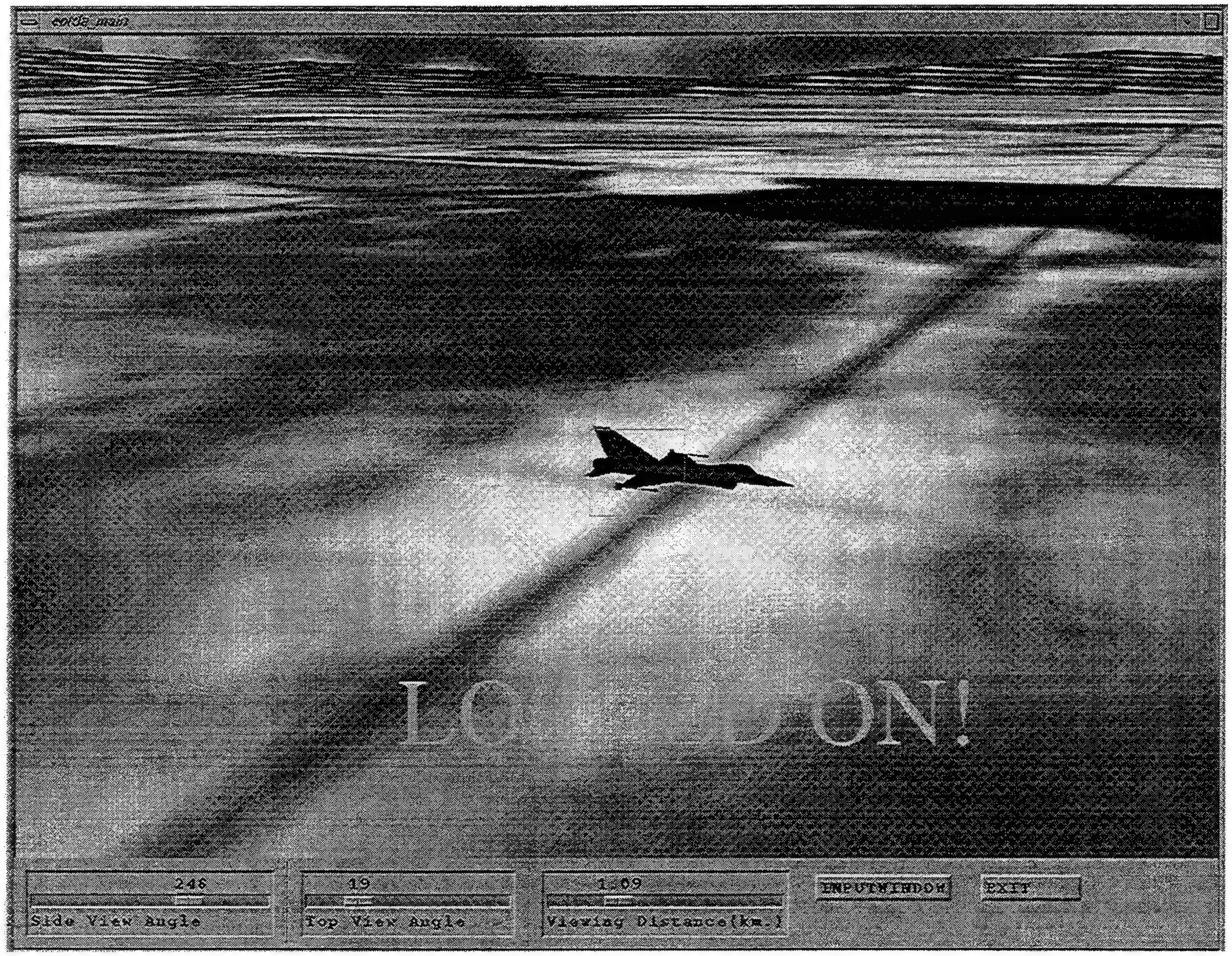
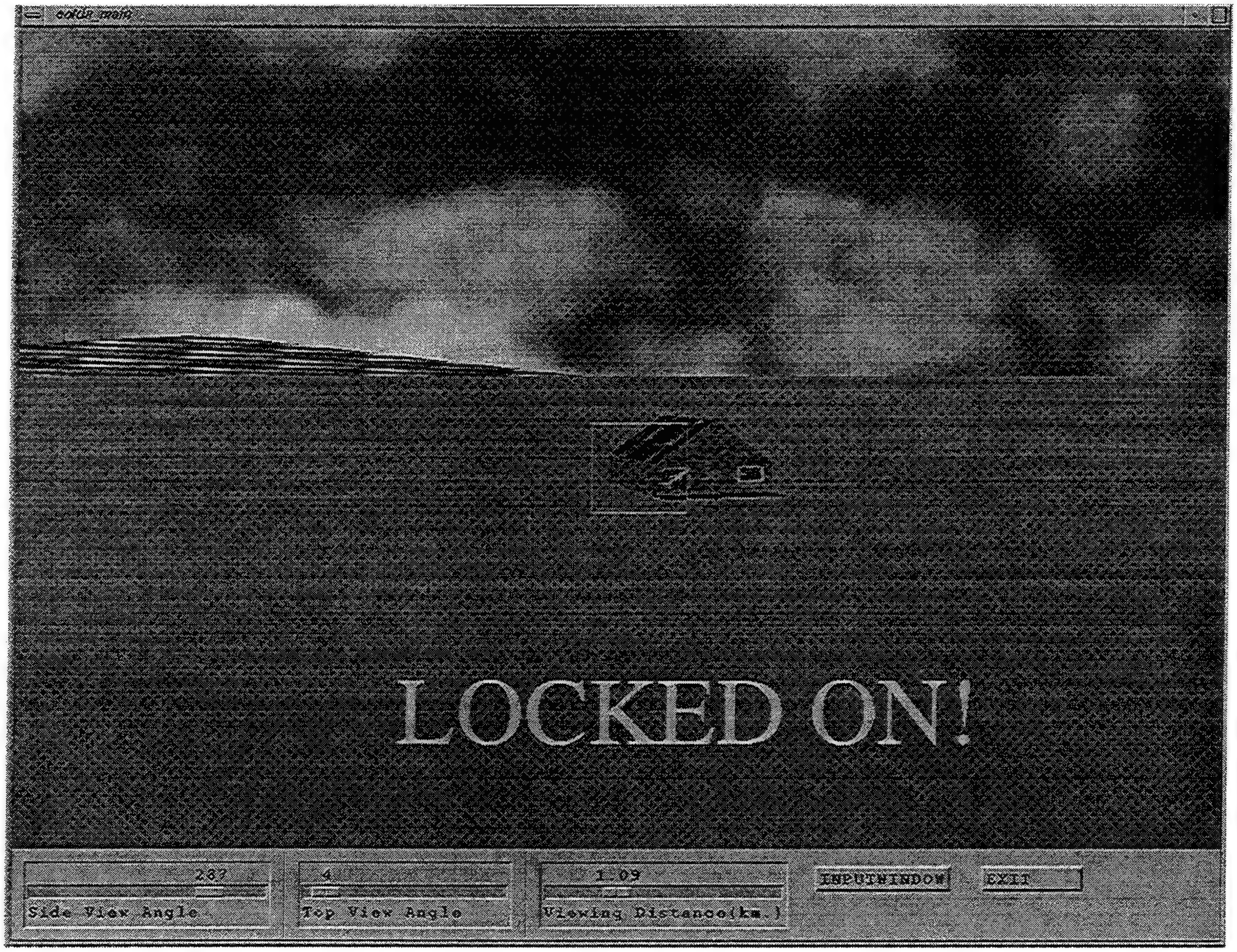


Figure 26. The TV Screen Output Of The Aircraft Target.



Side View Angle 267

Top View Angle 4

Viewing Distance(km.) 1.09

Figure 27. The TV Screen Output Of The Building Target.

APPENDIX B.

```
/***********************************************************************
* This is eotda_main.C                                                 *
*                                                                      *
*       This program is written to form a core program for EOTDA software and  *
*  utilizes the 3-D graphical target models.The program basically written     *
*  by using openGL programming language .The program consist of basically     *
*  three major parts.First part is related with the physical inputs and       *
*  calculations, second part is User Interface Design,and finally the third   *
*  part is related with displaying the graphical model based on the inputs    *
*  entered by the user.Since texturing costs a high computing time of CPU     *
*  this program is developed under Reality Engines.                           *
*                                                                      *
*                                                                      *
*                                                                      *
*       Authors :       Ltjg. Mustafa YILMAZ                           *
*                       Ltjg. Mehmet GORGULU                           *
*                                                                      *
***********************************************************************/


#include <iostream.h>           // C++ I/O subsystem..
#include <stdio.h>
#include <stdlib.h>                     // Get the exit() functiion definition
#include <math.h>
#include <fstream.h>
#include <stream.h>
#include <strstream.h>
#include <iomanip.h>
#include <Xm/MainW.h>
#include <Xm/Xm.h>                      // Get the Motif library......
#include <Xm/Form.h>                    // We are going to use a Form container widget.
#include <Xm/Frame.h>                   // Get the Frame widget.
#include <Xm/RowColumn.h>       // Get the RowColumn  widget.
#include <Xm/CascadeB.h>        // Get the CascadeButton widget.
#include <Xm/PushB.h>                   // Get the PushButton widget.
#include <Xm/PushBG.h>
#include <Xm/DialogS.h>         // DialogShell widgets
#include <Xm/Label.h>           // Label widgets
#include <Xm/LabelG.h>          // Label Gadets.
#include <Xm/Separator.h>               // Separator widgets
#include <Xm/Scale.h>                   //Get the Scale widget.
#include <Xm/MessageB.h>        // Get the MessageBox.
#include <Xm/Text.h>            // Get the Text Widget.
#include <Xm/TextF.h>                   // Get the Text Widget.
#include <Xm/ToggleBG.h>        // Get the ToggleBG Widget.

#include <Xm/ArrowBG.h>         // Get the ArrowBG Widget.
#include <GL/GLwMDrawA.h>       // We are going to use an OpenGL Motif Draw widget
#include <X11/StringDefs.h>
#include <X11/keysym.h>

#include <GL/gl.h>                      // Get the OpenGL required includes.
#include <GL/glu.h>
#include <GL/glx.h>
```

```
#include "materialsupport.h"        // Get material names.
#include "materialsupport_funcs.h"          // Get the material support functions.
#include "drawsupport_funcs.h"              // Get the drawing functions.
#include "texturesupport_funcs.h"           // Get the texture support functions.
#include "TextureDisplayList.h"
#include "eotda_funcs.h"                    // Get this program's function declarations.
#include "eotda_main.h"             // Get the variables and others for "main" program.
#include "global_targets.h"                 // Get the globals for this program.
#include "makeRasterFont.h"                 // Get the font support routines.
//#include "NPSimage.h"                     // Get the NPS image class definition.
#define __COMMON__
#include "eotda_globals.h"


winddata         wind ;                               // Create an object for winddata class
timedata Time_of_day ;             // Create an object for timedata class

static GLuint fontHandle1,fontHandle2;      // Base of font display lists.

char buffer1[10000] ,                             // Buffer for pixel red color values...(100x100)
     buffer2[10000] ;                       // Buffer for pixel luminance values...(100x100)
```

```
void main(int argc, char **argv)
{

  XmString message_string;          // Compound strings.

  Arg wargs[15];                                  // Args used with XtSetArg below.

 message_string = XmStringCreateLtoR ("Please Wait Initializing the Display Lists.", charset);

   // Fallback resources for the application.
   static GLbyte * fallback_resources[] = {
      "*form*background:            SGISlateBlue",  // /usr/lib/rgb.txt
      "*frame*shadowType:            SHADOW_IN",
      "*glwidget*width:            1400",
      "*glwidget*height:            900",
      "*glwidget*rgba:            TRUE",
      "*glwidget*doublebuffer:       TRUE",
      "*glwidget*allocateBackground: TRUE",
      "eotda_main*geometry: 1400x1000+0+0", // width, height, xoff, yoff
      "InputWindow*geometry:1400x1000+0+0",
      "info*geometry:10x10+800+800",
         NULL
   };


   // Create the top level widget.
   toplevel = XtAppInitialize(
      &app_context,                 // Application context.
      "eotda_main",                 // Application class.
      NULL, 0,                      // Command line option list.
      &argc, argv,                  // Command line args.
      fallback_resources,
      NULL,                                 // Argument list.
      0);                           // Number of arguments.

// Create a pop_up display window to inform the user during the initialization proccess of the
// program .

   n = 0;
   XtSetArg (wargs[n], XmNmessageString, message_string);  n++;
   info = XmCreateInformationDialog (toplevel,
                                        "Please Wait Initializing the Display Lists.", wargs, n);
  XtManageChild (info);

// Position the pop_up display in the center of the screen..
   n = 0;
   XtSetArg (wargs[n], XmNx, 200);  n++;
   XtSetArg (wargs[n], XmNy, 200);  n++;
   XtSetValues (XtParent(info), wargs, n);

// Get rid of compound strings.
   XmStringFree(message_string);

// Create Form widget. This is the top level container for below widgets.
```

```
   n = 0;
   XtSetArg(wargs[n],XmNfractionBase, 30); n++;
   MainWindow = XmCreateForm(toplevel, "MainWindow", wargs, n);
   XtManageChild(MainWindow);

 // Call the initializing function.This the first window we see when the program starts execution..
   BuildInputScreenDialog();


 // Create widget for  display.But do NOT manage  till "Display" button is pushed
   n = 0;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNbottomAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNbottomPosition, 27); n++;
   DisplayWindow = XmCreateForm(MainWindow,"DisplayWindow",wargs,n);


 // Create a frame that holds the display window
   n = 0;
   XtSetArg (wargs[n], XmNshadowThickness,2 ); n++;
   XtSetArg (wargs[n], XmNshadowType,XmSHADOW_IN); n++;
   MainDisplayFrame = XmCreateFrame (DisplayWindow, "MainDisplayFrame", wargs, n);
   XtManageChild (MainDisplayFrame);


 // Create Form widget. This is the  container widget for rowcolumn widgets.Do NOT manage
 // this widget either.Manage it when "Display" button is pressed...
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNtopPosition, 27); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNbottomAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   ControlWindow = XmCreateForm(MainWindow, "ControlWindow", wargs, n);
 // Create rowcolumn widget to display information.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNbottomAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg(wargs[n],XmNpacking, XmPACK_TIGHT); n++;
   rc = XmCreateRowColumn(ControlWindow,"rowcol",wargs,n);
   XtManageChild (rc);

 // Call the function that creates the Scales under the bulletin board and manages them..
   Create_The_Scales(rc) ;

 // Create a GL widget & initialize the z-buffer."XtSetArg(wargs[n], GLwNdepthSize,1); n++;"
 // is essential to initialize the z-buffer. Without this line z-buffering doesn't work and there will be // no hidden
 surface elemination...
```

```
   n = 0;
   XtSetArg(wargs[n], GLwNdepthSize,1); n++;
   glw = GLwCreateMDrawingArea(MainDisplayFrame, "glwidget", wargs, n);
   XtManageChild (glw);

// Add callbacks to the glw widget.
   XtAddCallback(glw, GLwNginitCallback,  initCB,   (XtPointer) NULL);
   XtAddCallback(glw, GLwNexposeCallback, exposeCB, (XtPointer) NULL);
   XtAddCallback(glw, GLwNresizeCallback, resizeCB, (XtPointer) NULL);
   XtAddCallback(glw, GLwNinputCallback, inputCB, (XtPointer) NULL);


// Add in the work procedure.A work procedure is called repeatedly whenever there are no
// events to process.
   workprocid = XtAppAddWorkProc(app_context, (XtWorkProc)drawWP,
                          (XtPointer)NULL);

// Instantiate it now the toplevel container widget.
  XtRealizeWidget(toplevel);

// Loop for events.
  XtAppMainLoop(app_context);

}
```

```
/*************************************************************************
*  Here we create all the widgets for input screen and manage the parent        *
* InputWindow widget so that we will see this screen first when the              *
* program is executed.When the "Display" button is pressed on this screen        *
* the InputWindow widget is unmanaged and DisplayWindow widget is managed        *
* so we will see the graphical display.                                           *
**************************************************************************/
void BuildInputScreenDialog()
{

        Widge  rc  ,        // Locally defined container widget.
               rc1 ,        // All other child widgets
               rc2 ,
               rc3 ,
               rc4 ;

        Arg wargs[15];              // Args used with XtSetArg below.

// Here is the global form that holds four different forms under it.
   n = 0;
   XtSetArg(wargs[n],XmNfractionBase, 40); n++;
   InputWindow = XmCreateForm(MainWindow, "InputWindow", wargs, n);
   XtManageChild(InputWindow);

// This the form that holds target related widgets under it.
   n = 0;
   XtSetArg(wargs[n],XmNfractionBase, 20); n++;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNrightPosition, 20); n++;
   XtSetArg(wargs[n],XmNbottomAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNbottomPosition, 20); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   Widget TargetForm = XmCreateForm(InputWindow, "TargetForm", wargs, n);
   XtManageChild(TargetForm);

// Create the target related widgets here..
  create_target_screen_widgets (TargetForm);

// This the form that holds sensor related widgets under it.
   n = 0;
   XtSetArg(wargs[n],XmNfractionBase, 20); n++;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNleftPosition, 22); n++;
   XtSetArg(wargs[n],XmNbottomAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNbottomPosition, 20); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   Widget SensorForm = XmCreateForm(InputWindow, "SensorForm", wargs, n);
   XtManageChild(SensorForm);

// Create the sensor related widgets here..
  create_sensor_screen_widgets (SensorForm);
// This the form that holds Background related widgets under it.
   n = 0;
   XtSetArg(wargs[n],XmNfractionBase, 40); n++;
   XtSetArg(wargs[n],XmNbottomAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNtopPosition, 20); n++;
   Widget BackgroundForm = XmCreateForm(InputWindow, "BackgroundForm", wargs, n);
   XtManageChild(BackgroundForm);

// Create the background related widgets here..
  create_background_screen_widgets (BackgroundForm);


}// End of  BuildInputScreenDialog()..
```

```
/************************************************************************
*         This function creates the necessary widgets for the target part           *
*    of the input_screen.                                                                   *
************************************************************************/
void create_target_screen_widgets(Widget parent)
{
  Widget          pulldown,          // Pulldown widget.
                  option;            // Cascade for the pulldown.



  XmString label_string;             // String.

  Arg wargs[10];                     // Same old args stuff.

 // This the rowcolumn that holds Target puldown widgets under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   Widget TargetRowColumn1 = XmCreateRowColumn(parent,
                                    "TargetRowColumn1", wargs, n);
   XtManageChild(TargetRowColumn1);



// Add a pulldown to the RowColumn.
  n = 0;
  pulldown = XmCreatePulldownMenu (TargetRowColumn1, "pulldown", wargs, n);


// Add the entries to the pulldown.
  make_pulldown_entry(pulldown, "TANK    ", targetCB, (XtPointer) &_TANK);
  make_pulldown_entry(pulldown, "TRUCK   ", targetCB, (XtPointer) &_TRUCK);
  make_pulldown_entry(pulldown, "SHIP    ", targetCB, (XtPointer) &_SHIP);
  make_pulldown_entry(pulldown, "HELO    ", targetCB, (XtPointer) &_HELO);
  make_pulldown_entry(pulldown, "BUILDING", targetCB, (XtPointer) &_HOUSE);
  make_pulldown_entry(pulldown, "AIRCRAFT", targetCB, (XtPointer) &_PLANE);

// Create the option menu that has the pulldown under it.
  label_string = XmStringCreateSimple("TARGET  :   ");
  n = 0;
  XtSetArg(wargs[n], XmNlabelString, label_string); n++;
  XtSetArg(wargs[n], XmNsubMenuId, pulldown);  n++;
  option = XmCreateOptionMenu (TargetRowColumn1, "optionMenu", wargs, n);

// Manage the label widget.
  XtManageChild (option);

// Free the string..
  XmStringFree(label_string);
```

```
// This the rowcolumn that holds Material puldown widgets under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNtopWidget,TargetRowColumn1 ); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   Widget TargetRowColumn11 = XmCreateRowColumn(parent,
                                        "TargetRowColumn11", wargs, n);
   XtManageChild(TargetRowColumn11);


// Add a pulldown to the RowColumn.
  n = 0;
  XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
  Widget pulldown2 = XmCreatePulldownMenu (TargetRowColumn11, "pulldown2", wargs, n);

// Add the entries to the pulldown.When material is included into the computations
// material names are to defined in global_targets.h file and used here as user data
// pased into the callback.
  make_pulldown_entry(pulldown2, "STEEL    ", materialCB, (XtPointer) &_TANK);
  make_pulldown_entry(pulldown2, "ALIMINIUM", materialCB, (XtPointer) &_TANK);
  make_pulldown_entry(pulldown2, "CONCERETE", materialCB, (XtPointer) &_TANK);
  make_pulldown_entry(pulldown2, "COMPOSITE", materialCB, (XtPointer) &_TANK);

// Create the option menu that has the pulldown under it.
  label_string = XmStringCreateSimple("MATERIAL:    ");
  n = 0;
  XtSetArg(wargs[n], XmNlabelString, label_string); n++;
  XtSetArg(wargs[n], XmNsubMenuId, pulldown2);  n++;
  Widget option2 = XmCreateOptionMenu (TargetRowColumn11, "optionMenu2", wargs, n);
  XtManageChild (option2);

// Free the string..
  XmStringFree(label_string);

// This the rowcolumn that holds TargetArea  widget under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNtopWidget,TargetRowColumn11 ); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   Widget TargetRowColumn2 = XmCreateRowColumn(parent,
                                        "TargetRowColumn2", wargs, n);
```

```
// This is the Target area gadget.
  XtVaCreateManagedWidget("Target Area (A):     ",xmLabelGadgetClass ,
                    TargetRowColumn2,XmNorientation, XmHORIZONTAL,NULL);

// Create the text widget for the Target Area.
  Widget TextWidget1 = XtVaCreateManagedWidget("TextWidget1",
                    xmTextWidgetClass ,TargetRowColumn2 ,NULL);

// Add Callbacks to the TextWidget1 widget.
  XtAddCallback(TextWidget1,XmNmodifyVerifyCallback,TargetAreaCB,NULL);

// Now manage the parent RowColumn Widget.
  XtManageChild(TargetRowColumn2);


// This the rowcolumn that holds TargetHeading  widget under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNtopWidget,TargetRowColumn2 ); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   TargetRowColumn3 = XmCreateRowColumn(parent,
                                    "TargetRowColumn3", wargs, n);

// This is the Target heading gadget.
  XtVaCreateManagedWidget("Target Heading :     ",xmLabelGadgetClass ,
                    TargetRowColumn3,XmNorientation, XmHORIZONTAL,NULL);

// Create the text widget for the TargetHeading....
  Widget TextWidget2 = XtVaCreateManagedWidget("TextWidget2",
                    xmTextWidgetClass ,TargetRowColumn3 ,NULL);

// Add Callbacks to the TextWidget2 widget.
  XtAddCallback(TextWidget2,XmNmodifyVerifyCallback,TargetHeadingCB,NULL);

// Now manage the parent RowColumn Widget.
  XtManageChild(TargetRowColumn3);

// This the rowcolumn that holds Target Speed  widget under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNtopWidget,TargetRowColumn3 ); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   TargetRowColumn4 = XmCreateRowColumn(parent,
                                    "TargetRowColumn4", wargs, n);
```

```
// This is the Target speed gadget.
   XtVaCreateManagedWidget("Target Speed (km/hr):",xmLabelGadgetClass ,
                    TargetRowColumn4,XmNorientation, XmHORIZONTAL,NULL);

// Create the text widget for the Target Speed.
   Widget TextWidget3 = XtVaCreateManagedWidget("TextWidget3",
                    xmTextWidgetClass ,TargetRowColumn4 ,NULL);

// Add Callbacks to the TextWidget3 widget.
   XtAddCallback(TextWidget3,XmNmodifyVerifyCallback,TargetSpeedCB,NULL);

// Now manage the parent RowColumn Widget.
   XtManageChild(TargetRowColumn4);


// Create the bulletin board  widget
    n=0;
    XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
    XtSetArg(wargs[n],XmNtopWidget,TargetRowColumn4 ); n++;
    XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
    TargetBB5= XmCreateBulletinBoard(parent, "TargetBB5", wargs, n);
    XtManageChild(TargetBB5);
// Create a Frame widget to make it so we can resize the window.
    n = 0;
    XtSetArg (wargs[n], XmNshadowThickness,2 ); n++;
    XtSetArg (wargs[n], XmNshadowType,XmSHADOW_OUT); n++;
    Widget TargetFrame5 = XmCreateFrame (TargetBB5, "TargetFrame5",
                                          wargs, n);
    XtManageChild (TargetFrame5);


// This the rowcolumn that holds Target Status  widgets under it.
    n = 0;
    XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
    XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
    XtSetArg (wargs[n], XmNradioBehavior, TRUE); n++;
    Widget TargetRowColumn5 = XmCreateRowColumn(TargetFrame5,
                                  "TargetRowColumn5", wargs, n);


// Create the label for Engine Status.
   n = 0;
   Widget Label = XtVaCreateManagedWidget("Engine Status :  ",xmLabelGadgetClass,
                    TargetRowColumn5,NULL);


// Here we create a Radio Button Gadget for Engine Status.
   Widget EngineOn = XtVaCreateManagedWidget("EngineOn",
                    xmToggleButtonGadgetClass ,TargetRowColumn5 ,NULL);

// Add Callbacks to the ON gadget.
   XtAddCallback(EngineOn,XmNvalueChangedCallback,EngineStatusCB,(XtPointer) 1);
```

```
// Here we create a Radio Button Gadget fot Targut Status.
  Widget EngineOff = XtVaCreateManagedWidget("EngineOff",
                   xmToggleButtonGadgetClass ,TargetRowColumn5 ,NULL);


// Add Callbacks to the OFF gadget.
  XtAddCallback(EngineOff,XmNvalueChangedCallback,EngineStatusCB,(XtPointer) 0);

// Now manage the parent RowColumn Widget.
  XtManageChild(TargetRowColumn5);




// Create the bulletin board  widget
   n=0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNtopWidget,TargetBB5 ); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   TargetBB6= XmCreateBulletinBoard(parent, "TargetBB6", wargs, n);
   XtManageChild(TargetBB6);

// Create a Frame widget
   n = 0;
   XtSetArg (wargs[n], XmNshadowThickness,2 ); n++;
   XtSetArg (wargs[n], XmNshadowType,XmSHADOW_OUT); n++;
   Widget TargetFrame6 = XmCreateFrame (TargetBB6, "TargetFrame6",
                                         wargs, n);
   XtManageChild (TargetFrame6);

// This the rowcolumn that holds Target Status  widgets under it.
   n = 0;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNradioBehavior, TRUE); n++;
   Widget TargetRowColumn6 = XmCreateRowColumn(TargetFrame6,
                                   "TargetRowColumn6", wargs, n);

// Create the label for Fire Status.
  n = 0;
  Widget Label2 = XtVaCreateManagedWidget("Fire Status   :  ",xmLabelGadgetClass,
                   TargetRowColumn6,NULL);


// Here we create a Radio Button Gadget fot Fire Status.
  Widget Fired = XtVaCreateManagedWidget("Fired ",
                   xmToggleButtonGadgetClass ,TargetRowColumn6 ,NULL);

// Add Callbacks to the ON gadget.
  XtAddCallback(Fired,XmNvalueChangedCallback,FireStatusCB,(XtPointer) 1);

// Here we create a Radio Button Gadget fot Targut Status.
```

```
  Widget NotFired = XtVaCreateManagedWidget("NotFired",
xmToggleButtonGadgetClass ,TargetRowColumn6 ,NULL);


// Add Callbacks to the OFF gadget.
  XtAddCallback(NotFired,XmNvalueChangedCallback,FireStatusCB,(XtPointer) 0);


// Now manage the parent RowColumn Widget.
  XtManageChild(TargetRowColumn6);


// Create the bulletin board  widget
   n=0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNtopWidget,TargetBB6 ); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   TargetBB7= XmCreateBulletinBoard(parent, "TargetBB7", wargs, n);

// Don't manage it here. Manage with respect to Target selected
// XtManageChild(TargetBB7);



// Create a Frame widget
   n = 0;
   XtSetArg (wargs[n], XmNshadowThickness,2 ); n++;
   XtSetArg (wargs[n], XmNshadowType,XmSHADOW_OUT); n++;
   Widget TargetFrame7 = XmCreateFrame (TargetBB7, "TargetFrame7",
                                        wargs, n);
   XtManageChild (TargetFrame7);


// This the rowcolumn that holds Target Status  widgets under it.
   n = 0;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNradioBehavior, TRUE); n++;
   Widget TargetRowColumn7 = XmCreateRowColumn(TargetFrame7,
                                "TargetRowColumn7", wargs, n);

// Create the label for Fire Status.
  n = 0;
  Widget Label3 = XtVaCreateManagedWidget("Heat Isolated:  ",
                  xmLabelGadgetClass,TargetRowColumn7,NULL);

// Here we create a Radio Button Gadget for Isolation Status.
  Widget Isolated = XtVaCreateManagedWidget("Yes ",
                  xmToggleButtonGadgetClass ,TargetRowColumn7 ,NULL);

// Add Callbacks to the Isolated gadget.
  XtAddCallback(Isolated,XmNvalueChangedCallback,IsolationStatusCB,
                  (XtPointer)1);
```

```
// Here we create a Radio Button Gadget fot Targut Status.
  Widget NotIsolated = XtVaCreateManagedWidget("No ",
                  xmToggleButtonGadgetClass ,TargetRowColumn7 ,NULL);


// Add Callbacks to the NotIsolated gadget.
  XtAddCallback(NotIsolated,XmNvalueChangedCallback,IsolationStatusCB,
                  (XtPointer)0);


// Now manage the parent RowColumn Widget.
  XtManageChild(TargetRowColumn7);


}   // End of create_target_screen_widgets
```

```
/*********************************************************************
*       This function creates the necessary widgets for the sensor part        *
* of the input_screen.                                                                                    *
*********************************************************************/
void create_sensor_screen_widgets (Widget parent)
{

  Arg wargs[10];                    // Same old args stuff.



// This the rowcolumn that holds Sensor puldown widgets under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
  // XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   Widget SensorRowColumn1 = XmCreateRowColumn(parent,
                                        "SensorRowColumn1", wargs, n);
   XtManageChild(SensorRowColumn1);

// Add a pulldown to the RowColumn.
  n = 0;
  Widget pulldown = XmCreatePulldownMenu (SensorRowColumn1, "pulldown", wargs, n);
  // Add the entries to the pulldown.
  make_pulldown_entry(pulldown, "IR      ", SensorCB, (XtPointer) &_IR);
  make_pulldown_entry(pulldown, "TV      ", SensorCB, (XtPointer) &_TV);
  make_pulldown_entry(pulldown, "LASER   ", SensorCB, (XtPointer) &_LASER);


// Create the option menu that has the pulldown under it.
  n = 0;
  XmString label_string = XmStringCreateSimple("SENSOR: ");
  XtSetArg(wargs[n], XmNlabelString, label_string); n++;
  XtSetArg(wargs[n], XmNsubMenuId, pulldown);  n++;
  Widget option = XmCreateOptionMenu (SensorRowColumn1, "optionMenu3", wargs, n);
  XtManageChild (option);
  XmStringFree(label_string);

// This the rowcolumn that holds Sensor puldown widgets under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_WIDGET); n++;
  // XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg(wargs[n],XmNleftWidget,SensorRowColumn1); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   SensorRC11 = XmCreateRowColumn(parent,
                                        "SensorRC11", wargs, n);
   XtManageChild(SensorRC11);
```

```
// Add a pulldown to the RowColumn.
  n = 0;
  Widget pulldown1 = XmCreatePulldownMenu (SensorRC11, "pulldown1", wargs, n);
  // Add the entries to the pulldown1.
  make_pulldown_entry(pulldown1, "TEMP.", IR_ScaleCB, (XtPointer)
                                                              &_TEMPERATURE_SCALE);
  make_pulldown_entry(pulldown1, "GRAY ", IR_ScaleCB, (XtPointer) &_GRAY_SCALE);
  make_pulldown_entry(pulldown1, "SYCL.", IR_ScaleCB, (XtPointer) &_SYCLIC_SCALE);


// Create the option menu that has the pulldown under it.
  n = 0;
  label_string = XmStringCreateSimple("IR_SCALE :");
  XtSetArg(wargs[n], XmNlabelString, label_string); n++;
  XtSetArg(wargs[n], XmNsubMenuId, pulldown1);  n++;
  Widget optionMenu1 = XmCreateOptionMenu (SensorRC11, "optionMenu1", wargs, n);
  XtManageChild (optionMenu1);
  XmStringFree(label_string);


// This the rowcolumn that holds Relative Aperture  widget under it.
  n = 0;
  XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
  XtSetArg(wargs[n],XmNtopWidget,SensorRowColumn1); n++;
  XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
  XtSetArg (wargs[n], XmNnumColumns, 1); n++;
  XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
  Widget SensorRowColumn2 = XmCreateRowColumn(parent,
                                    "SensorRowColumn2", wargs, n);



// This is the Relative Aperture gadget.
  XtVaCreateManagedWidget("Relative Aperture    :",xmLabelGadgetClass ,
                    SensorRowColumn2,XmNorientation, XmHORIZONTAL,NULL);

// Create the text widget for the Relative Aperture.
  Widget TextWidget1 = XtVaCreateManagedWidget("TextWidget1",
                    xmTextWidgetClass ,SensorRowColumn2 ,NULL);

// Add Callbacks to the TextWidget1 widget.
  XtAddCallback(TextWidget1,XmNmodifyVerifyCallback,RelativeApertureCB,NULL);

// Now manage the parent RowColumn Widget.
  XtManageChild(SensorRowColumn2);


// This the rowcolumn that holds Aperture Dram  widget under it.
  n = 0;
  XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
  XtSetArg(wargs[n],XmNtopWidget,SensorRowColumn2); n++;
  XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
  XtSetArg (wargs[n], XmNnumColumns, 1); n++;
  XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
  Widget SensorRowColumn3 = XmCreateRowColumn(parent,
                                    "SensorRowColumn3", wargs, n);

// This is the Relative Aperture gadget.
  XtVaCreateManagedWidget("Aperture Dram        :",xmLabelGadgetClass ,
                 SensorRowColumn3,XmNorientation, XmHORIZONTAL,NULL);

// Create the text widget for the Relative Aperture.
  Widget TextWidget2 = XtVaCreateManagedWidget("TextWidget2",
                 xmTextWidgetClass ,SensorRowColumn3 ,NULL);

// Add Callbacks to the TextWidget2 widget.
  XtAddCallback(TextWidget2,XmNmodifyVerifyCallback,ApertureDramCB,NULL);

// Now manage the parent RowColumn Widget.
  XtManageChild(SensorRowColumn3);


// This the rowcolumn that holds Magnification  widget under it.
  n = 0;
  XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
  XtSetArg(wargs[n],XmNtopWidget,SensorRowColumn3); n++;
  XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
  XtSetArg (wargs[n], XmNnumColumns, 1); n++;
  XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
  Widget SensorRowColumn4 = XmCreateRowColumn(parent,
                                    "SensorRowColumn4", wargs, n);

// This is the Relative Aperture gadget.
  XtVaCreateManagedWidget("Magnification        :",xmLabelGadgetClass ,
                 SensorRowColumn4,XmNorientation, XmHORIZONTAL,NULL);

// Create the text widget for the Magnification.
  Widget TextWidget3 = XtVaCreateManagedWidget("TextWidget3",
                 xmTextWidgetClass ,SensorRowColumn4 ,NULL);

// Add Callbacks to the TextWidget3 widget.
  XtAddCallback(TextWidget3,XmNmodifyVerifyCallback,MagnificationCB,NULL);

// Now manage the parent RowColumn Widget.
  XtManageChild(SensorRowColumn4);


// This the rowcolumn that holds Optical Transmittance  widget under it.
  n = 0;
  XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
  XtSetArg(wargs[n],XmNtopWidget,SensorRowColumn4); n++;
```

```
    XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
    XtSetArg (wargs[n], XmNnumColumns, 1); n++;
    XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
    Widget SensorRowColumn5 = XmCreateRowColumn(parent,
                                   "SensorRowColumn5", wargs, n);


 // This is the Relative Aperture gadget.
   XtVaCreateManagedWidget("Optical Transmittance :",xmLabelGadgetClass ,
               SensorRowColumn5,XmNorientation, XmHORIZONTAL,NULL);

 // Create the text widget for the Optical Transmittance.
   Widget TextWidget4 = XtVaCreateManagedWidget("TextWidget4",
               xmTextWidgetClass ,SensorRowColumn5 ,NULL);

 // Add Callbacks to the TextWidget4 widget.
   XtAddCallback(TextWidget4,XmNmodifyVerifyCallback,
               OpticalTransmittanceCB,NULL);

 // Now manage the parent RowColumn Widget.
   XtManageChild(SensorRowColumn5);


 // This the rowcolumn that holds Eln. Band Width  widget under it.
    n = 0;
    XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
    XtSetArg(wargs[n],XmNtopWidget,SensorRowColumn5); n++;
    XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
    XtSetArg (wargs[n], XmNnumColumns, 1); n++;
    XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
    Widget SensorRowColumn6 = XmCreateRowColumn(parent,
                                   "SensorRowColumn6", wargs, n);

 // This is the Relative Aperture gadget.
   XtVaCreateManagedWidget("Eln. Band Width       :",xmLabelGadgetClass ,
               SensorRowColumn6,XmNorientation, XmHORIZONTAL,NULL);

 // Create the text widget for the Eln. Band Width.
   Widget TextWidget5 = XtVaCreateManagedWidget("TextWidget5",
               xmTextWidgetClass ,SensorRowColumn6 ,NULL);

 // Add Callbacks to the TextWidget5 widget.
   XtAddCallback(TextWidget5,XmNmodifyVerifyCallback,
               ElectronicBandwidthCB,NULL);

 // Now manage the parent RowColumn Widget.
   XtManageChild(SensorRowColumn6);
```

```
// This the rowcolumn that holds Sensor Height  widget under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNtopWidget,SensorRowColumn6); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   Widget SensorRowColumn7 = XmCreateRowColumn(parent,
                                        "SensorRowColumn7", wargs, n);

// This is the Relative Aperture gadget.
  XtVaCreateManagedWidget("Sensor Height        :",xmLabelGadgetClass ,
                   SensorRowColumn7,XmNorientation, XmHORIZONTAL,NULL);
// Create the text widget for the Relative Aperture.
  Widget TextWidget6 = XtVaCreateManagedWidget("TextWidget6",
                   xmTextWidgetClass ,SensorRowColumn7 ,NULL);

// Add Callbacks to the TextWidget6 widget.
  XtAddCallback(TextWidget6,XmNmodifyVerifyCallback,SensorHeightCB,NULL);

// Now manage the parent RowColumn Widget.
  XtManageChild(SensorRowColumn7);


// This the rowcolumn that holds Detectivity  widget under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNtopWidget,SensorRowColumn7); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   Widget SensorRowColumn8 = XmCreateRowColumn(parent,
                                        "SensorRowColumn8", wargs, n);


// This is the Relative Aperture gadget.
  XtVaCreateManagedWidget("Detectivity          :",xmLabelGadgetClass ,
                   SensorRowColumn8,XmNorientation, XmHORIZONTAL,NULL);

// Create the text widget for the Detectivity.
  Widget TextWidget7 = XtVaCreateManagedWidget("TextWidget7",
                   xmTextWidgetClass ,SensorRowColumn8 ,NULL);

// Add Callbacks to the TextWidget7 widget.
  XtAddCallback(TextWidget7,XmNmodifyVerifyCallback,DetectivityCB,NULL);

// Now manage the parent RowColumn Widget.
  XtManageChild(SensorRowColumn8);
```

```
// This the rowcolumn that holds Detector Elements  widget under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNtopWidget,SensorRowColumn8); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   Widget SensorRowColumn9 = XmCreateRowColumn(parent,
                                        "SensorRowColumn9", wargs, n);

// This is the Relative Aperture gadget.
  XtVaCreateManagedWidget("Detector Elements    :",xmLabelGadgetClass ,
                    SensorRowColumn9,XmNorientation, XmHORIZONTAL,NULL);

// Create the text widget for the Detector Elements.
  Widget TextWidget8 = XtVaCreateManagedWidget("TextWidget8",
                    xmTextWidgetClass ,SensorRowColumn9 ,NULL);

// Add Callbacks to the TextWidget8 widget.
  XtAddCallback(TextWidget8,XmNmodifyVerifyCallback,DetectorElementsCB,NULL);

// Now manage the parent RowColumn Widget.
  XtManageChild(SensorRowColumn9);


// This the rowcolumn that holds Firing Angle  widget under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNtopWidget,SensorRowColumn9); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   Widget SensorRowColumn10 = XmCreateRowColumn(parent,
                                        "SensorRowColumn10", wargs, n);


// This is the Relative Aperture gadget.
  XtVaCreateManagedWidget("Firing Angle         :",xmLabelGadgetClass ,
                    SensorRowColumn10,XmNorientation, XmHORIZONTAL,NULL);

// Create the text widget for the Firing Angle.
  FiringAngleTextWidget = XtVaCreateManagedWidget("FiringAngleTextWidget",
                    xmTextWidgetClass ,SensorRowColumn10 ,XmNvalue,"10.0",NULL);

// Add Callbacks to the FiringAngleTextWidget widget.
  XtAddCallback(FiringAngleTextWidget,XmNmodifyVerifyCallback,FiringAngleCB,NULL);

// Now manage the parent RowColumn Widget.
  XtManageChild(SensorRowColumn10);
// This the rowcolumn that holds Relative Aperture  widget under it.
```

```
  n = 0;
  XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
  XtSetArg(wargs[n],XmNtopWidget,SensorRowColumn10); n++;
  XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
  XtSetArg (wargs[n], XmNnumColumns, 1); n++;
  XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
  Widget SensorRowColumn11 = XmCreateRowColumn(parent,
                                  "SensorRowColumn11", wargs, n);


// This is the Firing Distance gadget.
  XtVaCreateManagedWidget("Firing Distance      :",xmLabelGadgetClass ,
                 SensorRowColumn11,XmNorientation, XmHORIZONTAL,NULL);

// Create the text widget for the Firing Distance.
  FiringDistanceTextWidget = XtVaCreateManagedWidget("FiringDistanceTextWidget",
                 xmTextWidgetClass ,SensorRowColumn11,XmNvalue,"4000.0",NULL);

// Add Callbacks to the FiringDistanceTextWidget widget.
  XtAddCallback(FiringDistanceTextWidget,XmNmodifyVerifyCallback,FiringDistanceCB,
                                                                          NULL);

// Now manage the parent RowColumn Widget.
  XtManageChild(SensorRowColumn11);


}
```

```
/************************************************************************
*         This function creates the necessary widgets for the background         *
* part  of the input_screen.                                                       *
                                                                                   *
************************************************************************/

void create_background_screen_widgets (Widget parent)
{

  Widget            rowcolumn ,
                    button     ;
  XmString    label_string;
   Arg wargs[10];                    // Same old args stuff.


// This the form that holds Background puldown widgets under it.
   n = 0;
   XtSetArg(wargs[n],XmNfractionBase, 40); n++;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNbottomAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNrightPosition, 15); n++;
   XtSetArg(wargs[n],XmNbottomPosition, 30); n++;
   Widget Backgroundform1 = XmCreateForm(parent,
                                     "Backgroundform1", wargs, n);
   XtManageChild(Backgroundform1);


// This the rowcolumn that holds Pull down Menu  widget under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNtopPosition, 20); n++;
   XtSetArg(wargs[n],XmNleftPosition, 5); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   Widget BackgroundRowColumn1 = XmCreateRowColumn(Backgroundform1,
                                     "BackgroundRowColumn1", wargs, n);

// Add another pulldown to the RowColumn (for object selection).
  n = 0;
  Widget pulldown = XmCreatePulldownMenu (BackgroundRowColumn1,
                                                  "pulldown2", wargs, n);
// Add the entries to the pulldown.
  make_pulldown_entry(pulldown, "SOIL    ", Background1CB, (XtPointer) &_SOIL);
  make_pulldown_entry(pulldown, "GRASS   ", Background2CB, (XtPointer) &_GRASS);
  make_pulldown_entry(pulldown, "SEA     ", Background3CB, (XtPointer) &_SEA);
  make_pulldown_entry(pulldown, "ASPHALT ", Background4CB, (XtPointer) &_ASPHALT);

// Create the option menu that has the pulldown under it.
  n = 0;
```

```
   String string1 = "BACKGROUND:";
   label_string = XmStringCreate(string1);
   XtSetArg(wargs[n], XmNlabelString, label_string); n++;
   XtSetArg(wargs[n], XmNsubMenuId, pulldown);  n++;
   Widget option = XmCreateOptionMenu (BackgroundRowColumn1, "optionMenu2",
                                                           wargs, n);
  XtManageChild (option);


// Free the label_string....
 XmStringFree(label_string);

// Here manage the widget.
  XtManageChild (BackgroundRowColumn1);

// This the form that holds  Display widget under it.
   n = 0;
   XtSetArg(wargs[n],XmNfractionBase, 16); n++;
   XtSetArg(wargs[n],XmNbottomAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNtopPosition, 30); n++;
   Widget Backgroundform3 = XmCreateForm(parent,
                                "Backgroundform3", wargs, n);
   XtManageChild(Backgroundform3);

// Here define a rowcolumn widget for display.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNbottomAttachment, XmATTACH_POSITION); n++;
   XtSetArg(wargs[n],XmNtopPosition, 4); n++;
   XtSetArg(wargs[n],XmNleftPosition, 8); n++;
   XtSetArg(wargs[n],XmNrightPosition, 10); n++;
   XtSetArg(wargs[n],XmNbottomPosition, 12); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNnumColumns, 1); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   Widget BackgroundRC31 = XmCreateRowColumn(Backgroundform3,
                                "BackgroundRC31", wargs, n);

// Add an Display PushButton.
  label_string = XmStringCreateLtoR("DISPLAY", charset);
  n = 0;
  XtSetArg(wargs[n], XmNalignment, XmALIGNMENT_CENTER); n++;
  XtSetArg(wargs[n], XmNlabelString, label_string); n++;
  DisplayButton = XmCreatePushButton(BackgroundRC31, "Display", wargs, n);

// Initial callback is finished so we can manage Display button so the user can use.
  XtManageChild(DisplayButton);

  XmStringFree(label_string);
 // Now add an Display callback for the PushButton.
   XtAddCallback(DisplayButton, XmNarmCallback, DisplayCB, (XtPointer) NULL);

 // Now manage the parent RowColumn Widget.
   XtManageChild(BackgroundRC31);


 // This the form for Background  widgets..
    n = 0;
    XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNbottomAttachment, XmATTACH_POSITION); n++;
    XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_POSITION); n++;
    XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNleftPosition, 15); n++;
    XtSetArg(wargs[n],XmNbottomPosition, 30); n++;
    Widget Backgroundform2 = XmCreateForm(parent,
                                       "Backgroundform2", wargs, n);
    XtManageChild(Backgroundform2);



 // This the rowcolumn that holds Background  widgets under it.
    n = 0;
    XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
    XtSetArg (wargs[n], XmNnumColumns, 1); n++;
    XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
    Widget BackgroundRC3 = XmCreateRowColumn(Backgroundform2,
                                       "BackgroundRC3", wargs, n);

 // This is the Wind Speed label gadget.
   XtVaCreateManagedWidget("Wind Speed :",xmLabelGadgetClass ,
                 BackgroundRC3,XmNorientation, XmHORIZONTAL,NULL);

 // This is the label widget which holds the wind speed and it is modified when the wind speed is
 // changed.
   WindSpeedLabel =XtVaCreateManagedWidget("WindSpeedLabel",
                 xmLabelGadgetClass,BackgroundRC3,
                 XtVaTypedArg,XmNlabelString,XmRString,"0  ",3,
                 XmNuserData, &wind,
                 NULL);

 // Here we create an arrow to increment the wind velocity..
   Widget UpArrow = XtVaCreateManagedWidget("UpArrow",
                 xmArrowButtonGadgetClass,BackgroundRC3,
                 XmNarrowDirection, XmARROW_UP, NULL);

 // Add the callback for Up Arrow
   XtAddCallback (UpArrow, XmNarmCallback, ChangeWindSpeedCB, (XtPointer) 1);
```

```
// Here we create an arrow to decrement the wind velocity..
   Widget DownArrow = XtVaCreateManagedWidget("DownArrow",
                    xmArrowButtonGadgetClass,BackgroundRC3,
                    XmNarrowDirection, XmARROW_DOWN, NULL);

// Add the callback for Down Arrow
   XtAddCallback (DownArrow, XmNarmCallback, ChangeWindSpeedCB, (XtPointer) -1);

// Here manage the widget.
   XtManageChild (BackgroundRC3);

// Here define a rowcolumn widget for wind direction.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg(wargs[n],XmNtopWidget, BackgroundRC3); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg(wargs[n],XmNentryVerticalAlignment, XmALIGNMENT_CONTENTS_TOP);n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   Widget BackgroundRC4 = XmCreateRowColumn(Backgroundform2,
                                "BackgroundRC4", wargs, n);
// This is the Wind Direction gadget.
   XtVaCreateManagedWidget("Wind Dir :",xmLabelGadgetClass ,
                    BackgroundRC4,XmNorientation, XmHORIZONTAL,
                    NULL);

// Create the text widget for the WindDirection.
   Widget WindDirection = XtVaCreateManagedWidget("WindDirection",
                    xmTextWidgetClass ,BackgroundRC4 ,NULL);

// Add Callbacks to the WindDirection widget.
   XtAddCallback(WindDirection,XmNmodifyVerifyCallback,WindDirectionCB,
                    NULL);

// This is the Aeresol gadget.
   XtVaCreateManagedWidget(" Aeresol  :",xmLabelGadgetClass ,
                    BackgroundRC4,XmNorientation, XmHORIZONTAL,
                    NULL);

// Create the text widget for the Aeresol.
   Widget Aeresol = XtVaCreateManagedWidget("Aeresol",
                    xmTextWidgetClass ,BackgroundRC4 ,NULL);

// Add Callbacks to the Aeresol widget.
   XtAddCallback(Aeresol,XmNmodifyVerifyCallback,AeresolCB,
                    NULL);

// Now manage the parent RowColumn Widget.
   XtManageChild(BackgroundRC4);


// Here define a rowcolumn widget for wind direction.
```

```
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg(wargs[n],XmNtopWidget, BackgroundRC4); n++;
   XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg(wargs[n],XmNentryVerticalAlignment, XmALIGNMENT_CONTENTS_TOP);n++;
   Widget BackgroundRC41 = XmCreateRowColumn(Backgroundform2,
                                   "BackgroundRC41", wargs, n);

// This is the Lat/Long gadget.
  XtVaCreateManagedWidget("Latitude :",xmLabelGadgetClass ,
                 BackgroundRC41,XmNorientation, XmHORIZONTAL,
                 NULL);
// Create the text widget for the Latitude.
  Widget Latitude = XtVaCreateManagedWidget("Latitude",
                 xmTextWidgetClass ,BackgroundRC41 ,NULL);

// Add Callbacks to the Latitude widget.
  XtAddCallback(Latitude,XmNmodifyVerifyCallback,LatitudeCB,
                 NULL);


// This is the Lat/Long gadget.
  XtVaCreateManagedWidget(" Longitude:",xmLabelGadgetClass ,
                 BackgroundRC41,XmNorientation, XmHORIZONTAL,
                 NULL);

// Create the text widget for the Longitude.
  Widget Longitude = XtVaCreateManagedWidget("Longitude",
                 xmTextWidgetClass ,BackgroundRC41 ,NULL);

// Add Callbacks to the Latitude widget.
  XtAddCallback(Longitude,XmNmodifyVerifyCallback,LongitudeCB,
                 NULL);


// Here manage the widget.
  XtManageChild (BackgroundRC41);


// Create the bulletin board1  widget
   n=0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n],XmNtopWidget, BackgroundRC41); n++;
   Widget BB= XmCreateBulletinBoard(Backgroundform2, "BB", wargs, n);
   XtManageChild(BB);
```

```
// Create a Frame widget .
   n = 0;
   XtSetArg (wargs[n], XmNshadowThickness,2 ); n++;
   XtSetArg (wargs[n], XmNshadowType,XmSHADOW_OUT); n++;
   Widget BackgroundRC6frame = XmCreateFrame (BB, "BackgroundRC6frame", wargs, n);
   XtManageChild (BackgroundRC6frame);



// This the rowcolumn that holds Background  widgets under it.
   n = 0;
    XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
   XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
   Widget BackgroundRC6 = XmCreateRowColumn(BackgroundRC6frame,
                                        "BackgroundRC6", wargs, n);

// This is the Time Of The Day label gadget.
  XtVaCreateManagedWidget("Time Of The Day :",xmLabelGadgetClass ,
                    BackgroundRC6,XmNorientation, XmHORIZONTAL,NULL);

// This is the Hour label gadget.
  XtVaCreateManagedWidget("Hour :",xmLabelGadgetClass ,
                    BackgroundRC6,XmNorientation, XmHORIZONTAL,NULL);

// This is the label widget which holds the hourlabel.
  TimeLabelHour =XtVaCreateManagedWidget("TimeLabelHour",
                    xmLabelGadgetClass,BackgroundRC6,
                    XtVaTypedArg,XmNlabelString,XmRString,"1",2,
                    XmNuserData, &Time_of_day,
                    NULL);

// Here we create an arrow to increment the hour..
  Widget HourUpArrow = XtVaCreateManagedWidget("HourUpArrow",
                    xmArrowButtonGadgetClass,BackgroundRC6,
                    XmNarrowDirection, XmARROW_UP, NULL);

// Add the callback for Up Arrow
  XtAddCallback (HourUpArrow, XmNarmCallback, ChangeTimeHourCB, (XtPointer) 1);

// Here we create an arrow to decrement the hour.
  Widget HourDownArrow = XtVaCreateManagedWidget("HourDownArrow",
                    xmArrowButtonGadgetClass,BackgroundRC6,
                    XmNarrowDirection, XmARROW_DOWN, NULL);

// Add the callback for Down Arrow
  XtAddCallback (HourDownArrow, XmNarmCallback, ChangeTimeHourCB, (XtPointer) -1);

// This is the Month label gadget.
  XtVaCreateManagedWidget("  Month :",xmLabelGadgetClass ,
                    BackgroundRC6,XmNorientation, XmHORIZONTAL,NULL);

// This is the label widget which holds the Month label.
  TimeLabelMonth =XtVaCreateManagedWidget("TimeLabelMonth",
                    xmLabelGadgetClass,BackgroundRC6,
                    XtVaTypedArg,XmNlabelString,XmRString,"1",2,
                    XmNuserData, &Time_of_day,
                    NULL);

 // Here we create an arrow to increment the Month
   Widget MonthUpArrow = XtVaCreateManagedWidget("MonthUpArrow",
                    xmArrowButtonGadgetClass,BackgroundRC6,
                    XmNarrowDirection, XmARROW_UP, NULL);

 // Add the callback for Up Arrow
   XtAddCallback (MonthUpArrow, XmNarmCallback, ChangeTimeMonthCB, (XtPointer) 1);

 // Here we create an arrow to deccrement theMonth
   Widget MonthDownArrow = XtVaCreateManagedWidget("MonthDownArrow",
                    xmArrowButtonGadgetClass,BackgroundRC6,
                    XmNarrowDirection, XmARROW_DOWN, NULL);

 // Add the callback for Down Arrow
   XtAddCallback (MonthDownArrow, XmNarmCallback, ChangeTimeMonthCB, (XtPointer)-1);


 // This is the Year label gadget.
   XtVaCreateManagedWidget("  Year :",xmLabelGadgetClass ,
                    BackgroundRC6,XmNorientation, XmHORIZONTAL,NULL);

 // This is the label widget which holds the wind speed and it is modified
 // when the wind speed is changed.
   TimeLabelYear =XtVaCreateManagedWidget("TimeLabelYear",
                    xmLabelGadgetClass,BackgroundRC6,
                    XtVaTypedArg,XmNlabelString,XmRString,"1994",4,
                    XmNuserData, &Time_of_day,
                    NULL);
 // Here we create an arrow to increment the wind velocity..
   Widget YearUpArrow = XtVaCreateManagedWidget("YearUpArrow",
                    xmArrowButtonGadgetClass,BackgroundRC6,
                    XmNarrowDirection, XmARROW_UP, NULL);

 // Add the callback for Up Arrow
   XtAddCallback (YearUpArrow, XmNarmCallback, ChangeTimeYearCB, (XtPointer) 1);

 // Here we create an arrow to deccrement the Month.
   Widget YearDownArrow = XtVaCreateManagedWidget("MonthDownArrow",
                    xmArrowButtonGadgetClass,BackgroundRC6,
                    XmNarrowDirection, XmARROW_DOWN, NULL);

 // Add the callback for Down Arrow
   XtAddCallback (YearDownArrow, XmNarmCallback, ChangeTimeYearCB, (XtPointer) -1);
 // Here manage the widget.
   XtManageChild (BackgroundRC6);

 // This the rowcolumn that holds Pull down Menu  widget under it.
   n = 0;
   XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n],XmNleftAttachment, XmATTACH_FORM); n++;
```

```
    XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNtopWidget, BackgroundRC6); n++;
    XtSetArg(wargs[n],XmNorientation, XmHORIZONTAL); n++;
    XtSetArg (wargs[n], XmNnumColumns, 1); n++;
    XtSetArg (wargs[n], XmNpacking, XmPACK_TIGHT); n++;
    Widget BackgroundRC7 = XmCreateRowColumn(Backgroundform2,
                                    "BackgroundRC7", wargs, n);


// Add another pulldown to the RowColumn cloudiness sellection button
   n = 0;
   Widget pulldown10 = XmCreatePulldownMenu (BackgroundRC7,
                                              "pulldown10", wargs, n);

// Add the entries to the pulldown10.
   make_pulldown_entry(pulldown10, "OVERCAST    ", CloudinessCB, (XtPointer) &_SOIL);
   make_pulldown_entry(pulldown10, "CLOUDY   ", CloudinessCB, (XtPointer) &_GRASS);
   make_pulldown_entry(pulldown10, "CLEAR    ", CloudinessCB, (XtPointer) &_SEA);

// Create the option menu that has the pulldown under it.
   n = 0;
   label_string = XmStringCreateSimple("CLOUDINESS:");
   XtSetArg(wargs[n], XmNlabelString, label_string); n++;
   XtSetArg(wargs[n], XmNsubMenuId, pulldown10);  n++;
   Widget option10 = XmCreateOptionMenu (BackgroundRC7, "optionMenu10",
                                                       wargs, n);
   XmStringFree(label_string);

// Here manage the widget.
   XtManageChild (option10);

// This is the FOG DENSITY gadget.
   XtVaCreateManagedWidget("FOG DENSITY:",xmLabelGadgetClass ,
                  BackgroundRC7,XmNorientation, XmHORIZONTAL,NULL);

// Create a Frame widget
    n = 0;
    XtSetArg(wargs[n],XmNtopAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNrightAttachment, XmATTACH_FORM); n++;
    XtSetArg(wargs[n],XmNbottomAttachment, XmATTACH_FORM); n++;
    XtSetArg (wargs[n], XmNshadowThickness,2 ); n++;
    XtSetArg (wargs[n], XmNshadowType,XmSHADOW_OUT); n++;
    Widget FogFrame = XmCreateFrame (BackgroundRC7, "FogFrame", wargs, n);
    XtManageChild (FogFrame);

// Crate the fog scale
  n = 0 ;
  XtSetArg (wargs[n], XmNshowValue, True);        n++;
  XtSetArg (wargs[n], XmNorientation, XmHORIZONTAL); n++;
  XtSetArg (wargs[n], XmNminimum, 0); n++;
  XtSetArg (wargs[n], XmNmaximum, 100); n++;
  XtSetArg (wargs[n], XmNprocessingDirection, XmMAX_ON_RIGHT); n++;
  Widget Fogscale = XmCreateScale(FogFrame, "Fogscale", wargs, n);
  XtManageChild (Fogscale);
```

```
// Add the callbacks
  XtAddCallback (Fogscale, XmNvalueChangedCallback, FogDensityCB , (XtPointer) NULL);
  XtAddCallback (Fogscale, XmNdragCallback,         FogDensityCB , (XtPointer) NULL);

// Here manage the widget.
   XtManageChild (BackgroundRC7);


}// End of  create_background_screen_widgets(..) ....
```

```
/****************************************************************************
*                          make_pulldown_entry -                             *
* The goal of this function is to simplify the construction of individual entries in the   *
* pulldown menu. The routine assumes that a menubar has already been created and a   *
* menupane created for that menubar.                                         *
****************************************************************************/

void make_pulldown_entry(

  Widget menupane,                          // A menupane of the menubar.

  char *entrytext,                          // Display text for the pulldown entry.

  XtCallbackProc callback,                  // Name of procedure to call
                                            // when this menu entry is selected.

  XtPointer user_data)                  // Data to be sent the callback.

{

   Arg wargs[10];                           // Same old args stuff.

   Widget button;                           // Temp to hold the widget (not managed).

   // Specify the character set (set up XmStrings).
   XmStringCharSet charset = (XmStringCharSet) XmSTRING_DEFAULT_CHARSET;


 // Create this particular entry in the pulldown menu.
   n = 0;
   XtSetArg(wargs[n], XmNlabelString,XmStringCreateLtoR(entrytext, charset)); n++;
   button = XmCreatePushButton (menupane, entrytext, wargs, n);

// We are assuming no data structure needs to be passed in to the callback.
   XtAddCallback (button, XmNactivateCallback, callback, user_data);

// Manage the entry in the menupane.
   XtManageChild (button);

}
```

```
void Background1CB(Widget, XtPointer, XtPointer)
{
 // Call the display list for soil texturing
          BACKGROUND = Index + 3 ;
          glCallList(BACKGROUND) ;

}

void Background2CB(Widget, XtPointer, XtPointer)
{
 // Call the display list for grass texturing
         BACKGROUND = Index + 1 ;
         glCallList(BACKGROUND ) ;


}

void Background3CB(Widget, XtPointer, XtPointer)
{
 // Call the display list for sea texturing
          BACKGROUND = Index  ;
         glCallList(BACKGROUND ) ;
}


void Background4CB(Widget, XtPointer user_data, XtPointer)
{

 // Call the display list for asphalt texturing
         BACKGROUND = Index + 2 ;
         glCallList(BACKGROUND ) ;
}
```

```
void SensorCB(Widget, XtPointer user_data, XtPointer)
{
          float     Temp,mu,total_radiance;
// First Load the Sensor ID to SENSOR variable.
          int *SENSOR_NO = (int *)user_data ;
          SENSOR = *SENSOR_NO ;

// If the sensor is IR then we must modify the color table.
 if(SENSOR == _IR )
 {
// Manage IR_scale widget so color sellection button will be visible..
   XtManageChild(SensorRC11);

// First Find the extinction cooficient value.....
          mu = Calculate_Extinction();

// Now modify the color table with current Firing Distance.
          for(int index = 0 ; index < VERTICE_NUM ; index++)
          {
                    Temp = Original_Temp[index][0]   ;
                    total_radiance = Find_Total_Radiance(Temp);
                    Temp = Temp_At_Sensor(total_radiance,mu);
                    Temperature[index][0] = Temp  ;
                    Fill_Up_Color_Table(index,MODE);
          }//End of for loop..
} // End of if clause..

if(SENSOR == _TV )
 {

// Manage IR_scale widget so it will  be  not displayed anymore
   XtUnmanageChild(SensorRC11);

// Modify the environment color to dark gray.
          for(int j = 0 ; j < 2500 ; j++)
          {
                    Color [j][0] = 0.7 ;
                    Color [j][1] = 0.7 ;
                    Color [j][2] = 0.7 ;
                    Color [j][3] = 1.0 ;
          } //End of for loop.
  } // End of if(TV)
  if(SENSOR == _LASER ){// If color scale was displayed make it unvisible..
                                                            XtUnmanageChild(SensorRC11); }
 } //End of SensorCB......
 // This callback is called when the target option menu is called.

 void targetCB(Widget, XtPointer user_data, XtPointer )
 {
         float     Temp,mu,total_radiance;
//Load the target ID.
         int *TARGET_NO = (int *)user_data ;
         Target = *TARGET_NO ;


// If the current sensor is IR then we must modify the color table.
 if(SENSOR == _IR )
 {

// First Find the mu value.....
         mu = Calculate_Extinction();

// Now modify the color table with previous Firing Distance.
         for(int index = 0 ; index < VERTICE_NUM ; index++)
         {
                 Temp = Original_Temp[index][0]   ;
                 total_radiance = Find_Total_Radiance(Temp);
                 Temp = Temp_At_Sensor(total_radiance,mu);
                 Temperature[index][0] = Temp  ;
                 Fill_Up_Color_Table(index,MODE);
         }//End of for loop..
} // End of if clause..

if(SENSOR == _TV )
 {
         for(int j = 0 ; j < 2500 ; j++)
         {
                 Color [j][0] = 0.7 ;
                 Color [j][1] = 0.7 ;
                 Color [j][2] = 0.7 ;
                 Color [j][3] = 1.0;

         } //End of for loop.

 }
```

```
// Here we are going to open the related target file with according to the
// target selected by the user, and call the initilization function.
        switch(Target)
        {
                case TANK:
                FillInInitialTargetArrays("TankInitialFile.dat");
                XtManageChild(TargetRowColumn3);
                XtManageChild(TargetRowColumn4);
                XtManageChild(TargetBB5);
                XtManageChild(TargetBB6);
                XtUnmanageChild(TargetBB7);
                BACKGROUND = Index + 3 ;
                glCallList(BACKGROUND) ;
                break ;

                case TRUCK:   //This is not yet implemented...
                //FillInInitialTargetArrays("TruckInitialFile.dat");
                XtManageChild(TargetRowColumn3);
                XtManageChild(TargetRowColumn4);
                XtManageChild(TargetBB5);
                XtManageChild(TargetBB6);
                XtUnmanageChild(TargetBB7);
                BACKGROUND = Index + 3 ;
                glCallList(BACKGROUND) ;
                break ;

                case SHIP :
                FillInInitialTargetArrays("ShipInitialFile.dat");
                XtManageChild(TargetRowColumn3);
                XtManageChild(TargetRowColumn4);
                XtManageChild(TargetBB5);
                XtManageChild(TargetBB6);
                XtUnmanageChild(TargetBB7);
                BACKGROUND = Index  ;
                glCallList(BACKGROUND) ;
                break ;

                case HELO : // This target is also not yet implemented.
                //FillInInitialTargetArrays("HeloInitialFile.dat");
                XtManageChild(TargetRowColumn3);
                XtManageChild(TargetRowColumn4);
                XtManageChild(TargetBB5);
                XtManageChild(TargetBB6);
                XtUnmanageChild(TargetBB7);
                BACKGROUND = Index + 3 ;
                glCallList(BACKGROUND) ;
                break ;


                case HOUSE :
                FillInInitialTargetArrays("BuildingInitialFile.dat");
                XtUnmanageChild(TargetRowColumn3);
                XtUnmanageChild(TargetRowColumn4);
                XtUnmanageChild(TargetBB5);
                XtUnmanageChild(TargetBB6);
                XtManageChild(TargetBB7);
                BACKGROUND = Index + 3 ;
                glCallList(BACKGROUND) ;
                break ;


                case PLANE :
                FillInInitialTargetArrays("AircraftInitialFile.dat");
                XtManageChild(TargetRowColumn3);
                XtManageChild(TargetRowColumn4);
                XtManageChild(TargetBB5);
                XtManageChild(TargetBB6);
                XtUnmanageChild(TargetBB7);
                BACKGROUND = Index + 10 ;
                glCallList(BACKGROUND) ;
                break ;


                default :
                break ;
        }

}//End of targetCB....
```

```
/***********************************************************************
* This is the Display Callback.This callback activates the display and          *
* ControlWindow widgets , and Unmanage the InputWindow widget.                   *
***********************************************************************/
void DisplayCB(Widget, XtPointer, XtPointer)
{

        float     Temp,mu,total_radiance;

// Remove the InputWindow widget so it will be unvisible
   XtUnmanageChild (InputWindow);

// Manage DisplayWindow Widget Make it visible.
  XtManageChild (DisplayWindow);

// Manage MainWindow Widget. Make this window visible too.
  XtManageChild(ControlWindow);

// Store the targets vertice number in VERTICE_NUM so when we are modifying the color arrays
// we don't try to modify the unnacessary part of the array.
switch(Target)
        {
                case TANK:
                VERTICE_NUM =        Tank_Index ;
                break ;

                case TRUCK:
                //VERTICE_NUM =      Truck_Index ;
                break ;

                case SHIP :
                VERTICE_NUM =        Ship_Index ;
                break ;

                case HELO :
                //VERTICE_NUM =  Helo_Index ;
                break ;

                case HOUSE :
                VERTICE_NUM =        Building_Index ;
                break ;

                case PLANE :
                VERTICE_NUM =        Airplane_Index ;
                break ;
        }// End of switch statement
// If the sensor is IR then we must modify the color table.
 if(SENSOR == _IR )
 {
 // First Find the Extinction coeficient value.....
        mu = Calculate_Extinction();

// Now modify the color table with previous Firing Distance.
        for(int index = 0 ; index < VERTICE_NUM ; index++)
```

```
          {

                    Temp = Original_Temp[index][0]   ;
                    total_radiance = Find_Total_Radiance(Temp);
                    Temp = Temp_At_Sensor(total_radiance,mu);
                    Temperature[index][0] = Temp  ;
                    Fill_Up_Color_Table(index,MODE);

          }//End of for loop..
} // End of if clause..

// Here reset the state control variables.
DISPLAY_ENABLED = 1;
SEARCHED_ONES = 0 ;
READ_ONES = 0;
OLD_MAX_X = 1;
OLD_MAX_Y = 1;
MAX_X = 0;
MAX_Y = 0;
MAX_RED = 0.0;
ROW = 0;
COLOMN = 900 ;
Max_Contrast_Of_Scene = 0 ;



}// End of Display CallBack..
```

```
/**************************************************************************
*These following callbacks are not used in the program but since the interface has already *
* been designed the necessary callbacs are defined but no action has been taken.They kept *
* for future use.                                                          *
**************************************************************************/
static void materialCB(Widget, XtPointer , XtPointer){;}
static void TargetAreaCB(Widget, XtPointer , XtPointer){;}
static void TargetHeadingCB(Widget, XtPointer , XtPointer){;}
static void TargetSpeedCB(Widget, XtPointer , XtPointer){;}
static void EngineStatusCB(Widget, XtPointer , XtPointer){;}
static void FireStatusCB(Widget, XtPointer , XtPointer){;}
static void IsolationStatusCB(Widget, XtPointer , XtPointer){;}
static void RelativeApertureCB(Widget, XtPointer , XtPointer){;}
static void ApertureDramCB(Widget, XtPointer , XtPointer){;}
static void MagnificationCB(Widget, XtPointer , XtPointer){;}
static void OpticalTransmittanceCB(Widget, XtPointer , XtPointer){;}
static void ElectronicBandwidthCB(Widget, XtPointer , XtPointer){;}
static void SensorHeightCB(Widget, XtPointer , XtPointer){;}
static void DetectivityCB(Widget, XtPointer , XtPointer){;}
static void DetectorElementsCB(Widget, XtPointer , XtPointer){;}
static void FiringAngleCB(Widget, XtPointer , XtPointer){;}
static void FiringDistanceCB(Widget, XtPointer , XtPointer){;}
static void WindDirectionCB(Widget, XtPointer , XtPointer){;}
static void AeresolCB(Widget, XtPointer , XtPointer){;}
static void LongitudeCB(Widget, XtPointer , XtPointer){;}
static void LatitudeCB(Widget, XtPointer , XtPointer){;}
static void CloudinessCB(Widget, XtPointer , XtPointer){;}

/**************************************************************************
* This is the Change Wind Speed  Callback.
**************************************************************************/
static void ChangeWindSpeedCB(Widget, XtPointer user_data, XtPointer call_data)
{
        int incr = (int) user_data ;
        winddata          *Wind ;
        char buf[8];
        XtVaGetValues (WindSpeedLabel,XmNuserData,&Wind,NULL);
        Wind->value += incr ;
        if(Wind->value < 0) Wind->value = 0 ;
        sprintf(buf,"%d",Wind->value);
        XtVaSetValues(WindSpeedLabel,
                    XtVaTypedArg,XmNlabelString,XmRString,buf,strlen(buf),
                    NULL);
}


/**************************************************************************
* This is the Change Change Time Hour Callback.
**************************************************************************/
static void ChangeTimeHourCB(Widget, XtPointer user_data, XtPointer)
{

        int incr = (int) user_data ;
        timedata*TIME ;
        char buf[8];
        XtVaGetValues (TimeLabelHour,XmNuserData,&TIME,NULL);
        TIME->hour += incr ;
        if(TIME->hour < 1) TIME->hour = 1 ;
        if(TIME->hour > 24) TIME->hour = 24 ;
        sprintf(buf,"%d",TIME->hour);
        XtVaSetValues(TimeLabelHour,
                    XtVaTypedArg,XmNlabelString,XmRString,buf,2,
                    NULL);
        HOUR = TIME->hour ;

}

/*************************************************************************
 * This is the Change Time Month Callback.
 *************************************************************************/
static void ChangeTimeMonthCB(Widget, XtPointer user_data, XtPointer)
{

        int incr = (int) user_data ;
        timedata *TIME ;
        char buf[8];
        XtVaGetValues (TimeLabelMonth,XmNuserData,&TIME,NULL);
        TIME->month += incr ;
        if(TIME->month < 1) TIME->month = 1 ;
        if(TIME->month > 31) TIME->month = 31 ;
        sprintf(buf,"%d",TIME->month);
        XtVaSetValues(TimeLabelMonth,
                    XtVaTypedArg,XmNlabelString,XmRString,buf,2,
                    NULL);


}

/*************************************************************************
 * This is the Change Time Year Callback.
 *************************************************************************/
static void ChangeTimeYearCB(Widget, XtPointer user_data, XtPointer)
{

        int incr = (int) user_data ;
        timedata *TIME ;
        char buf[8];
        XtVaGetValues (TimeLabelYear,XmNuserData,&TIME,NULL);
        TIME->year += incr ;
        if(TIME->year < 1994) TIME->year = 1994 ;
        if(TIME->year > 2100) TIME->year = 2100 ;
        sprintf(buf,"%d",TIME->year);
        XtVaSetValues(TimeLabelYear,
                    XtVaTypedArg,XmNlabelString,XmRString,buf,4,
                    NULL);

}
```

```
/****************************************************************************
* This is the Fog  Density Callback. The scale value is divided to 30000 to make the
* density smaller so the fog color doesn't cover the whole screen.
****************************************************************************/
static void FogDensityCB(Widget, XtPointer , XtPointer call_data)
{
        XmScaleCallbackStruct * call_value = (XmScaleCallbackStruct *) call_data;
        density = (call_value -> value)/30000.0 ;
}


/****************************************************************************
* This is the IR Scale Callback. With the help of "Mode" variable the color that is going to
* loaded onto the targets are desided.
****************************************************************************/
static void IR_ScaleCB(Widget, XtPointer user_data, XtPointer)
{
        int *IR_SCALE_NO = (int *)user_data ;
        MODE = *IR_SCALE_NO ;
}
```

```
/************************************************************************
 * Here we Create_The_Scales () for the display screen.There are three scales     *
 * created.                                                                        *
 ************************************************************************/
void Create_The_Scales(Widget parent)

{
  Widget         pulldown,                    // Row coloumn widget
                 option   ,
                 button   ;


         Arg wargs[16];              // Args used with XtSetArg below.
         XmFontList fontlist;                  // Font List

         XmString titleString1 ,     //Titles for scales..
                  titleString2 ,
                  titleString3 ;

         XFontStruct *font;                    // Ptr to a font structure.

         XmString label_string;




// Create a compound string for the scale title
  font = XLoadQueryFont (XtDisplay (toplevel), "spc12x12c");
  fontlist = XmStringCreateFontList (font, charset);
  titleString1 = XmStringCreateLtoR ("Side View Angle      ", charset);
  titleString2 = XmStringCreateLtoR ("Top View Angle    ", charset);
  titleString3 = XmStringCreateLtoR ("Viewing Distance(km.)", charset);

// Create the bulletin board  widget
   n=0;
   XtSetArg (wargs[n], XmNbottomAttachment, XmATTACH_FORM); n++;
   XtSetArg (wargs[n], XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetArg (wargs[n], XmNbottomOffset, 2); n++;
   bulletin_board1= XmCreateBulletinBoard(parent, "bulletin1", wargs, n);
   XtManageChild(bulletin_board1);


// Create a Frame widget to make it so we can see the size of the window.
    n = 0;
    XtSetArg (wargs[n], XmNshadowThickness,2 ); n++;
    XtSetArg (wargs[n], XmNshadowType,XmSHADOW_IN); n++;
    Widget frame1 = XmCreateFrame (bulletin_board1, "frame1", wargs, n);
    XtManageChild (frame1);

// Set up arglist and create the scale1
  n = 0;
  XtSetArg (wargs[n], XmNfontList, fontlist);         n++;
  XtSetArg (wargs[n], XmNshowValue, True);            n++;
 XtSetArg (wargs[n], XmNtitleString, titleString1); n++;
 XtSetArg (wargs[n], XmNorientation, XmHORIZONTAL); n++;
 XtSetArg (wargs[n], XmNmaximum, 360); n++;
 XtSetArg (wargs[n], XmNprocessingDirection, XmMAX_ON_RIGHT); n++;
 scale1 = XmCreateScale(frame1, "scale1", wargs, n);
 XtManageChild (scale1);

// Add the callbacks
 XtAddCallback (scale1, XmNvalueChangedCallback, rotateyCB , (XtPointer) NULL);
 XtAddCallback (scale1, XmNdragCallback,        rotateyCB , (XtPointer) NULL);


// Add a separator
  n = 0;
  XtSetArg (wargs[n], XmNorientation, XmVERTICAL); n++;
  sep = XmCreateSeparator(parent,"separator",wargs,n);
  XtManageChild(sep);

// Create the bulletin board2  widget
  n=0;
  XtSetArg (wargs[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (wargs[n], XmNleftWidget,bulletin_board1 ); n++;
  XtSetArg (wargs[n], XmNbottomAttachment, XmATTACH_FORM); n++;
  XtSetArg (wargs[n], XmNbottomOffset, 2); n++;
  bulletin_board2= XmCreateBulletinBoard(parent, "bulletin2", wargs, n);
  XtManageChild(bulletin_board2);

// Create a Frame widget
   n = 0;
   XtSetArg (wargs[n], XmNshadowThickness,2 ); n++;
   XtSetArg (wargs[n], XmNshadowType,XmSHADOW_IN); n++;
   frame2 = XmCreateFrame (bulletin_board2, "frame2", wargs, n);
   XtManageChild (frame2);

// Set up arglist and create the scale2
  n = 0;
 XtSetArg (wargs[n], XmNfontList, fontlist);        n++;
 XtSetArg (wargs[n], XmNshowValue, True);           n++;
 XtSetArg (wargs[n], XmNtitleString, titleString2); n++;
 XtSetArg (wargs[n], XmNorientation, XmHORIZONTAL); n++;
 XtSetArg (wargs[n], XmNmaximum, 90); n++;
 XtSetArg (wargs[n], XmNminimum, 0); n++;
 XtSetArg (wargs[n], XmNprocessingDirection, XmMAX_ON_RIGHT); n++;
 scale2 = XmCreateScale(frame2, "scale2", wargs, n);
 XtManageChild (scale2);

// Here we set the initial value of this scale since it is different than
// minvalue of ths scale.
 XmScaleSetValue(scale2 ,int(Firing_Angle) );

// Add the callbacks
 XtAddCallback (scale2, XmNvalueChangedCallback, rotatexCB, (XtPointer) NULL);
 XtAddCallback (scale2, XmNdragCallback,        rotatexCB, (XtPointer) NULL);
```

```
// Add a separator
  n = 0;
  XtSetArg (wargs[n], XmNorientation, XmVERTICAL); n++;
  sep = XmCreateSeparator(parent,"separator",wargs,n);
  XtManageChild(sep);


// Create the bulletin board3  widget
  n=0;
  XtSetArg (wargs[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (wargs[n], XmNleftWidget,bulletin_board2 ); n++;
  XtSetArg (wargs[n], XmNbottomAttachment, XmATTACH_FORM); n++;
  XtSetArg (wargs[n], XmNbottomOffset, 2); n++;
  bulletin_board3= XmCreateBulletinBoard(parent, "bulletin3", wargs, n);
  XtManageChild(bulletin_board3);


// Create a Frame widget to make it so we can resize the window.
  n = 0;
  XtSetArg (wargs[n], XmNshadowThickness,2 ); n++;
  XtSetArg (wargs[n], XmNshadowType,XmSHADOW_IN); n++;
  Widget frame3 = XmCreateFrame (bulletin_board3, "frame3", wargs, n);
  XtManageChild (frame3);

// Set up arglist and create the scale3
  n = 0;
  XtSetArg (wargs[n], XmNfontList, fontlist);         n++;
  XtSetArg (wargs[n], XmNshowValue, True);            n++;
  XtSetArg (wargs[n], XmNtitleString, titleString3); n++;
 XtSetArg (wargs[n], XmNorientation, XmHORIZONTAL); n++;
 XtSetArg (wargs[n], XmNminimum, 0); n++;
 XtSetArg (wargs[n], XmNvalue, 400); n++;
 XtSetArg (wargs[n], XmNmaximum, 400); n++;
 XtSetArg (wargs[n], XmNdecimalPoints, 2); n++;
 XtSetArg (wargs[n], XmNprocessingDirection, XmMAX_ON_RIGHT); n++;
 scale3 = XmCreateScale(frame3, "scale3", wargs, n);
 XtManageChild (scale3);

 // Add the callbacks
 XtAddCallback (scale3, XmNvalueChangedCallback, translatezCB, (XtPointer) NULL);
 XtAddCallback (scale3, XmNdragCallback,        translatezCB, (XtPointer) NULL);

  // Here we set the initial value of this scale since it is different than minvalue of this scale.
  XmScaleSetValue(scale3 ,400 );

  // Get rid of the fontlists
  XmFontListFree (fontlist);

  // Get rid of the Stringlists.
  XmStringFree (titleString1);
  XmStringFree (titleString2);
  XmStringFree (titleString3);
  // Create widget to hold the push buttons..
  n = 0;
  XtSetArg (wargs[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (wargs[n], XmNleftWidget,bulletin_board3); n++;
  XtSetArg (wargs[n], XmNnumColumns, 1); n++;
  XtSetArg (wargs[n], XmNorientation, XmVERTICAL); n++;
  rc2 = XmCreateRowColumn(parent,"degree",wargs,n);
  XtManageChild(rc2);


// Create the bulletin board  widget for  INPUTWINDOW push button.
  n=0;
  BboardForInput= XmCreateBulletinBoard(rc2, "BboardForInput", wargs, n);
  XtManageChild(BboardForInput);

 // Now add an Input PushButton.
  label_string = XmStringCreateLtoR("INPUTWINDOW", charset);
  n = 0;
  XtSetArg(wargs[n], XmNalignment, XmALIGNMENT_CENTER); n++;
  XtSetArg(wargs[n], XmNlabelString, label_string); n++;
  button = XmCreatePushButton(BboardForInput, "InputScreen", wargs, n);
  XtManageChild(button);
  XmStringFree(label_string);

  // Now add an Exit callback for the PushButton.
  XtAddCallback(button, XmNarmCallback, BackToInputScreenCB, (XtPointer) NULL);

  // Create widget to hold the push buttons..
  n = 0;
  XtSetArg (wargs[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (wargs[n], XmNleftWidget,rc2 ); n++;
  XtSetArg (wargs[n], XmNnumColumns, 1); n++;
  XtSetArg (wargs[n], XmNorientation, XmVERTICAL); n++;
  Widget rc3 = XmCreateRowColumn(parent,"degree",wargs,n);
  XtManageChild(rc3);

// Create the bulletin board  widget for  Exit push button.
  n=0;
  BboardForExit= XmCreateBulletinBoard(rc3, "BboardForExit", wargs, n);
  XtManageChild(BboardForExit);

 // Now add an Exit PushButton.
  label_string = XmStringCreateLtoR("EXIT     ", charset);
  n = 0;
  XtSetArg(wargs[n], XmNalignment, XmALIGNMENT_CENTER); n++;
  XtSetArg(wargs[n], XmNlabelString, label_string); n++;
  button = XmCreatePushButton(BboardForExit, "exit", wargs, n);
  XtManageChild(button);
  XmStringFree(label_string);

 // Now add an Exit callback for the PushButton.
  XtAddCallback(button, XmNarmCallback, quitCB, (XtPointer) NULL);}
// End of the create the scales.......
```

```
/*****************************************************************************
*         This initial calback where it iscalled when the program reaches the definition of the      *
*"glw" display widget. What this callback does is it initializes the window routines               *
* display lists for texturing and the cube which is used for sky polygons. It also loads the        *
* initial values of some GL and user defined variables.                                              *
*****************************************************************************/
static void initCB (Widget w, XtPointer, XtPointer call_data)
{
// Get us a GLwDrawingAreaCallbackStruct ptr to the call_data.
   GLwDrawingAreaCallbackStruct *glptr = (GLwDrawingAreaCallbackStruct *) call_data;

   Arg wargs[1];      // Arg temp.

   XVisualInfo *vi;   // Pointer to XVisualInfo.

   GLint gdtmp;       // Used in the get capability stuff below.



   // Get the visual info...
   XtSetArg(wargs[0], GLwNvisualInfo, &vi);
   XtGetValues(w, wargs, 1);

   // Create a new GLX rendering context.
   // GL_TRUE -> Specify direct connection to graphics system (if possible).
   glx_context = glXCreateContext(XtDisplay(w), vi, 0, GL_TRUE);

   // Set the global window and display.
   global_display = XtDisplay(w);
   global_window  = XtWindow(w);

   // Make this drawing area the current one.
   GLwDrawingAreaMakeCurrent(w, glx_context);

   // Test to see if this machine has a z-buffer.
   if((glGetIntegerv(GL_DEPTH_BITS, &gdtmp), gdtmp) == 0)
   {
      cerr <<  "This machine does not have a hardware zbuffer" << endl;
      exit(0);
   }

// Turn on z-buffering.
   glEnable(GL_DEPTH_TEST);


// Turn on Gouraud shading.
   glShadeModel(GL_SMOOTH);



// Get a font we can use to display our messages. We usebthe first font to display "LOCK ON"
// and the second one to display numbers on the color scale...
   fontHandle1 = makeRasterFont(w, "-*-times-medium-r-normal--100-*-*-*-p-84-iso8859-1");
```

```
    fontHandle2 = makeRasterFont(w, "-*-times-medium-r-normal--14-*-*-*-p-84-iso8859-1");

// Set the Viewport for the first draw of the window.
    glViewport (0, 0, glptr->width, glptr->height);

// Set the initial viewing parameters.
    set_initial_viewing_values();

// Load the default target arrays here
   FillInInitialTargetArrays("TankInitialFile.dat");

// Call the display list to form the texturing list
   CallDisplayListForTexturing();


// Make a display List for Cube.
   CallDisplayListForCube();

// Load the default initial values to the variables .....
   MOUNTAIN = Index + 6 ;
   CLOUDS = Index + 5 ;
   BACKGROUND = Index+3  ;


// Set the texture environment we want.
    setGLTextureEnvironment();

// Unmanage the info screen so it wnt be visiuble any more...
  XtUnmanageChild (info);

}  // end of initCB()
```

```
/*************************************************************************
 *  exposeCB -                                                           *
 *  This routine is called whenever the window is uncovered.             *
 *************************************************************************/
static void exposeCB(Widget w, XtPointer, XtPointer )
{

// Set the window into which GL drawing should be done.
   GLwDrawingAreaMakeCurrent(w, glx_context);
  if(DISPLAY_ENABLED){ draw_the_scene();  } // otherwise we are in the input screen so
                                                       // don't spent time by calling display window.

}



/*************************************************************************
 *  resizeCB -                                                           *
 *  This routine is called whenever the window is moved or resized.      *
 *************************************************************************/
static void resizeCB (Widget w, XtPointer, XtPointer call_data )
{

   // Get us a GlwDrawingAreaCallbackStruct ptr to the call_data.
   GLwDrawingAreaCallbackStruct *glptr = (GLwDrawingAreaCallbackStruct *) call_data;


   // Set the window into which GL drawing should be done.
   GLwDrawingAreaMakeCurrent(w, glx_context);

   // Set the viewport using the window size currently set.
   glViewport (0, 0, (GLsizei) glptr->width, (GLsizei) glptr->height);

  if(DISPLAY_ENABLED){ draw_the_scene();  } // otherwise we are in the input screen so
                                                       // don't spent time by calling display window.
}
```

```
/*************************************************************************
*   inputCB -                                                              *
*  This routine handles all types of input from the GL widget.             *
*   In this particular example, the KeyRelease handles the                 *
*   Escape-Key so that its release exits the program.                      *
*************************************************************************/

static void inputCB (Widget, XtPointer, XtPointer call_data )
{

   // Get us a GLwDrawingAreaCallbackStruct ptr to the call_data.
   GLwDrawingAreaCallbackStruct *glptr = (GLwDrawingAreaCallbackStruct *) call_data;

   char ascii[1];              // Hold for the ascii chars returned from XLookupString.

   int nchars;      // Number of characters returned from XLookupString.

   KeySym keysym;   // The X version of the returned character.

   XKeyEvent *ptr;          // ptr to the Key Event structure.


   // We look at the type of event to see if we should pay attention...
   // In this example, we only pay attention to key release events.
   switch(glptr->event->type)
   {

     case KeyRelease:
       // We must convert the keycode to a KeySym before it is possible
       // to check if it is an escape. We also dump the Ascii into the
       // array ascii. The return value from XLookupString is the
       // number of characters dumped into array ascii.
       ptr = (XKeyEvent *) glptr->event;
       nchars = XLookupString(ptr, ascii, 1, &keysym, NULL);

       if(nchars == 1 && keysym == (KeySym) XK_Escape)
       {
         // We have an escape. Time to exit.
         exit(0);
       }

       break;



// This part of the program kept in the program forseeing that in the future some functions will be
// assigned to the  mouse buttons.
     case ButtonPress:
       // We have a button press from the mouse.
       // Which mouse button was it?
       switch(glptr->event->xbutton.button)
       {
         case Button2:  // The middle button was pressed.
```

```
            break;
          case Button1:   // The left button was pressed.
            break;
          case Button3:   // The right button was pressed.
            break;

          default:
            break;

        }  // end switch on which mouse button it was.
        break;

      case ButtonRelease:
        // We have a button release from the mouse.
        // Which mouse button was it?
        switch(glptr->event->xbutton.button)
        {
          case Button2:  // The middle button was pressed.
            break;
          case Button1:  // The left button was pressed.
            break;
          case Button3:  // The right button was pressed.
            break;

          default:
            break;

        }  // end switch on which mouse button it was.
        break;

      case MotionNotify:
        // We have motion on the mouse. We only get notified of motion when a mouse button is
       // pressed.
        break;
      default:
        break;
    }  // end of switch on event->type.
}  // end of inputCB.
/*******************************************************************************
*   quitCB -                                                                   *
*  This function is called whenever the "Quit" menu option     is selected.   *
*******************************************************************************/
static void quitCB (Widget w, XtPointer, XtPointer)
{
   // Bye, bye...
   XtCloseDisplay(XtDisplay(w));
   exit(0);

}  // End of quitCB

/*******************************************************************************
*   rotatexCB                                                                  *
*  This function is called whenever the Top view scale is moved.              *
*******************************************************************************/
```

```
static void rotatexCB(Widget, XtPointer , XtPointer call_data)
{

        float     Temp,mu,total_radiance,Extincted_temp;
        XmScaleCallbackStruct * call_value = (XmScaleCallbackStruct *) call_data;

        pitch = call_value -> value ;
        Firing_Angle = pitch;

 if(SENSOR == _IR )
 {

        // First Find the mu value.....
        mu = Calculate_Extinction();


        // Now modify the color table with previous Firing Angle.
        for(int index = 0 ; index < VERTICE_NUM ; index++)
        {
                Temp = Original_Temp[index][0]   ;
                total_radiance = Find_Total_Radiance(Temp);
                Temp = Temp_At_Sensor(total_radiance,mu);
                Temperature[index][0] = Temp  ;
                Fill_Up_Color_Table(index,MODE);

        }//End of for loop..

 } // End of if clause..

// Modify the statre control variables.....
SEARCHED_ONES = 0 ;
READ_ONES = 0;
ROW = 0;
COLOMN = 900 ;
OLD_MAX_X = MAX_X ;
OLD_MAX_Y = MAX_Y ;
MAX_RED = 0.0;
Max_Contrast_Of_Scene = 0  ;

} // End of rotatexCB....


/**************************************************************************
*   rotateyCB
*  This function is called whenever the Side view scale is moved.
**************************************************************************/
static void rotateyCB(Widget, XtPointer , XtPointer call_data)
{

        XmScaleCallbackStruct * call_value = (XmScaleCallbackStruct *) call_data;

        pitch = call_value -> value;

}
```

```
/*****************************************************************************
*   translatezCB                                                             *
*  This function is called whenever the Distance scale is moved.             *
*****************************************************************************/

static void translatezCB(Widget, XtPointer , XtPointer call_data)
{

        float     Temp,mu,kt,total_radiance;
        XmScaleCallbackStruct * call_value = (XmScaleCallbackStruct *) call_data;

        tz = call_value -> value ;
        Firing_Distance = tz ;

// If the sensor is IR then we must modify the color table.
 if(SENSOR == _IR ) {

        // First Find the mu value.....
        mu = Calculate_Extinction();

        // Now modify the color table with previous Firing Distance.
        for(int index = 0 ; index < VERTICE_NUM ; index++)
        {
                Temp = Original_Temp[index][0]   ;
                total_radiance = Find_Total_Radiance(Temp);
                Temp = Temp_At_Sensor(total_radiance,mu);
                Temperature[index][0] = Temp  ;
                Fill_Up_Color_Table(index,MODE);

        }//End of for loop..

} // End of if clause..


// Modify the state control variables....
SEARCHED_ONES = 0 ;
READ_ONES = 0;
ROW = 0;
COLOMN = 900 ;
OLD_MAX_X = MAX_X ;
OLD_MAX_Y = MAX_Y ;
MAX_RED = 0.0;
Max_Contrast_Of_Scene = 0 ;

}//End of translatezCB...

*****************************************************************************
*  drawWP -                                                                 *
*  This function is called by the work procedure. It is called repeatedly whenever there are    *
* no events to process. This function returns FALSE so that the work procedure does NOT      *
* stop calling it.                                                           *
*****************************************************************************/
```

```
GLboolean drawWP()
{
  if(DISPLAY_ENABLED){draw_the_scene(); }

   // If we do not return FALSE, the work procedure will stop calling this function.
   return(FALSE);

}
```

```
/******************************************************************************
 * draw_the_scene                                                             *
 *   This function draws the picture.                                         *
 ******************************************************************************/
void draw_the_scene()
{

// Turn on z-buffering!
  glEnable(GL_DEPTH_TEST);


  // Set the background color to cyan (RGBA) and clear the z-buffer.
  glClearColor(0.25, 0.0, 1.0, 1.0);
  glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);



  // Set the projection  matrix.
  // The near and far values are distances from the viewer
  // to the near (1.0) and far (10000.0) clipping planes.
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(45.0,  1.25,  1.0,  18000.0);


  // Load the ModelView matrix with a unit matrix.
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();


   if(SENSOR == _TV )
  {

  // Turn on the Light Model ...
  turnOnTheLightModel();

  // Turn on the Lights ...
  turnOnTheLights();
  //Enable the fog for the target.
    EnableFog(density) ;

  }


  //Here we check the position of the viewer..
  load_the_viewpoint();

  // We are at the viewpoint and looking towards
  // the reference point of the object.
  // Up vector is the vector orienting our view volume around
  // the line of sight.
  gluLookAt(Viewpoint[0], Viewpoint[1], Viewpoint[2],    // View point.
        Refpoint[0] , Refpoint[1] , Refpoint[2] ,       // Ref point, point we are looking towards.
        0.0, 1.0, 0.0 );                                // Up vector.
```

```
// Concatenate onto the ModelView matrix the results of our
// accumulative transformations.
// The coordinate passed in is the point about which
// we would like our operations (rotations/scales) to occur.
// Think of this point as the center of the displayed object.

   concatenate_accumulative_matrices(0.0,0.0,0.0 );


//First draw the target.That is the nearest object to the user.
    drawTarget(Target,0.0,-2.0,0.0);

   if(SENSOR == _TV )
   {
    // Draw the target shadow ..
    drawTargetShadow(Target,0.0,-2.0,0.0);

    // Turn on texturing ...
    glEnable(GL_TEXTURE_2D);

   }


// Here we are drawing the ground and calling the ground texturing.
        glCallList(BACKGROUND) ;
        drawGround(0.0, -2.0, 0.0,BACKGROUND,MODE);

// Here we are drawing the mountains and calling the mountain texturing.
        glCallList(MOUNTAIN) ;
        DrawTheMoutains(MODE);


//Call the display list for clouds texture and Cube.
        glCallList(CLOUDS) ;
        glCallList(CUBE) ;

//Here we draw the sun.The position of the sun is time dependent.
     DrawTheSun(HOUR);


if(SENSOR == _TV )
{
   //Turn off everything being turned on.(Save energy.....)
   glDisable(GL_TEXTURE_2D);
   DisableFog() ;
   turnOffTheLights();
   turnOffTheLightModel();
   Call_TV_Functions();

}//End of if(TV)..
```

```
if(SENSOR == _IR ) { Call_IR_Functions();}

// Swap the buffers as we are doing double buffering.
glXSwapBuffers(global_display, global_window);

}   // end of draw_the_scene
```

```
/*****************************************************************************
* The following routine actually does the matrix multiplication to compute        *
* the following formula.                                                          *
*                                                                                 *
* P' = P . T(to origin) . S(acc) . R(x acc) . R(y acc) . R(z acc)                 *
*        . T(to acc. loc) . T(back to specified center) . . lookat() .perspective()  *
*                                                                                 *
* The input coordinate is the point about which operations should be done.       *
* Think of that point as the point translated to the origin.                      *
*****************************************************************************/
void concatenate_accumulative_matrices(float refx, float refy, float refz)
{
  // Since the operation is done by multiplying the matrices,the order
  // of operation is important.

  // We are going to use the ModelView stack.
  glMatrixMode(GL_MODELVIEW);

  // translate center of object back to original location.
  glTranslatef(refx, refy, refz);
  glRotatef((pitch + 1.0), 1.0, 0.0, 0.0);          // Rotate to look from the top
  glRotatef(heading, 0.0, 1.0, 0.0);                // Rotate to lok from each side..

  // translate center of object to the origin.
  glTranslatef(-refx, -refy, -refz);

}


/**************************************************************************
* The following routine loads a unit matrix into the input array.                 *
**************************************************************************/
void unit(GLfloat *m)
{

  static GLfloat un[4][4] = { 1.0, 0.0, 0.0, 0.0,
                              0.0, 1.0, 0.0, 0.0,
                              0.0, 0.0, 1.0, 0.0,
                              0.0, 0.0, 0.0, 1.0 };

  long i,j;

  GLfloat *ptr;   // Get a temp ptr.


  ptr = m;  // Get a ptr to the start of m.

  // copy the matrix elements...
  for(i=0; i < 4; i=i+1)
  {
    for(j=0; j < 4; j=j+1)
    {
      *ptr++ = un[j][i];
    }
```

```
   }

 }// End oj unit(..)


/*************************************************************************
 * This function sets the initial viewing parameters.                    *
 *************************************************************************/
void set_initial_viewing_values()
{

  // Set the initial angles.
    heading = 0.0;
    pitch  = Firing_Angle;
    tz     = Firing_Distance ;
    roll   = 0.0;   //This value is not used but saved for future implementations which can
                               // be modified in rotatezCB(....)
    density = 0.0 ;//This value is used for fog implimentation

         Viewpoint[0] = 0.0 ;
         Viewpoint[1] = 0.0 ;
         Viewpoint[2] = tz+5;  // We don't want get in the targets which is at the origin..

         // We are loo,ing to the center..
         Refpoint[0] = 0.0 ;
         Refpoint[1] = 0.0 ;
         Refpoint[2] = 0.0 ;


}

/*************************************************************************
* This function sets the initial viewing parameters.                          *
*************************************************************************/
void load_the_viewpoint()
{ Viewpoint[0] = 0.0 ;  Viewpoint[1] = 0.0 ;  Viewpoint[2] = tz+5;}

/*************************************************************************
* This callback is used to return back to input screen from the display       *
* screen. This callback manages the InputScreen widget, and Unmanage the   *
* DisplayScreen and ControlScreen widgets.                                          *
*************************************************************************/
static void BackToInputScreenCB(Widget, XtPointer, XtPointer)
{

// Remove the DisplayWindow widget
   XtUnmanageChild (DisplayWindow);

// Remove the ControlWindow widget
   XtUnmanageChild (ControlWindow);

// Manage MainWindow Widget
   XtManageChild(InputWindow);
```

```
// Here set the global DISPLAY_ENABLED variable to zero so when draw the
// scene is called no function is executed..
DISPLAY_ENABLED = 0;

}

/***********************************************************************
*  This function reads-in the target data files and loads the vertex ,temperature and color *
*  arrays. Function return type is integer for error checking purposes.                     *
***********************************************************************/

int  FillInInitialTargetArrays(char * InputFile)
{

        int Vertice_no ;
        float      Vertex_x ,
                   Vertex_y ,
                   Vertex_z ,
                   Temp     ,
                   kt,
                   mu,
                   total_radiance = 0.0;
        const int size = 100 ;
        char buffer[size] ;                         // Used to store unnecassary characters.

        ifstream from  (InputFile);                 // Input Data File
        if (!from)
        {
        cout << " I can't open "<< InputFile << "File " << endl ;
        return 0;
        }

//  Since this function is called when a target is initialized find the "mu" value according to either
// default or sellected values of firin angle.
  mu = Calculate_Extinction();

while (!from.eof())
{
        from >> Vertice_no ;
        from >> Vertex_x ;
        from >> Vertex_y ;
        from >> Vertex_z ;
        from >> Temp     ;
        from.getline(buffer,size,'\n');             // Go to next line
        Temp = Temp  ;
        Original_Temp[Vertice_no][0]     = Temp  ;
        Vertex[Vertice_no][0]                  = Vertex_x ;
        Vertex[Vertice_no][1]                  = Vertex_y ;
        Vertex[Vertice_no][2]                  = Vertex_z ;

  // We read the temperature values then calculate theradiance of the target for this temperature
 //  value assuming the target is a black body radiator.
```

```
        kt=Boltzman_Constant*(Temp);
        total_radiance = Find_Total_Radiance(Temp);

  // Now with this radiance and mu extinction coef. find the equilant temp on the sensor.This
  // function applies necessary extinction to the signal and after finding the new level of the signal
  // it transforms it to temperature value again.This calculated temp. is loaded to the current temp.
  // array.
        Temp = Temp_At_Sensor(total_radiance,mu);
        Temperature[Vertice_no][0]= Temp ;

// Here the temperature values transformed to the equilant color values, and color array is loaded.
        Fill_Up_Color_Table(Vertice_no,MODE);


} // end while...

from.close();
 return 1;
}// End of function...



/*********************************************************************************
* This function calculte the extinction coeficient with respect to the angle that comes from    *
* " Top View Angle " scale. We used the complimentary angle for this calculation                *
* because the top view  is 90 degree for scale but 0 degree for LOWTRAN (LT.)codes              *
* This extinction values can be found directly running the LT. itself but it is very            *
* time consuming so we couldn't afford to spend that much time for a simulation that the        *
* time is very critical      . Instead we calculated the extinction coefficient by running LT.  *
* with five degree  increments and used those values here.                                      *
*********************************************************************************/

float Calculate_Extinction()
{

        float App_Angle= 90.0 - Firing_Angle ;

        if (App_Angle <= 90 && App_Angle > 85) return  1.1660 ;
        if (App_Angle <= 85 && App_Angle > 80) return  0.8099 ;
        if (App_Angle <= 80 && App_Angle > 75) return  0.6134 ;
        if (App_Angle <= 75 && App_Angle > 70) return  0.4650 ;
        if (App_Angle <= 70 && App_Angle > 65) return  0.3518 ;
        if (App_Angle <= 65 && App_Angle > 60) return  0.2702 ;
        if (App_Angle <= 60 && App_Angle > 55) return  0.2123 ;
        if (App_Angle <= 55 && App_Angle > 50) return  0.1251 ;
        if (App_Angle <= 50 && App_Angle > 45) return  0.1239 ;
        if (App_Angle <= 45 && App_Angle > 40) return  0.1217 ;
        if (App_Angle <= 40 && App_Angle > 35) return  0.1068 ;
        if (App_Angle <= 35 && App_Angle > 30) return  0.0960 ;
        if (App_Angle <= 30 && App_Angle > 25) return  0.0877 ;
        if (App_Angle <= 25 && App_Angle > 20) return  0.0816 ;
        if (App_Angle <= 20 && App_Angle > 15) return  0.0739 ;
        if (App_Angle <= 15 && App_Angle > 10) return  0.0739 ;
        if (App_Angle <= 10 && App_Angle >  5) return  0.0716 ;
        if (App_Angle <=  5 && App_Angle >  0)  return  0.0703 ;
        if (App_Angle == 0)                     return  0.0703 ;

}


/***********************************************************************
* This function finds the temperature equilant color for three different coloring schemes*
* and loads the color array.                                                              *
***********************************************************************/

int Fill_Up_Color_Table(int Vertice_no,int COLOR_MODE)
{

double red ;

        if(Temperature[Vertice_no][0] <= 10)        // Dark_Blue

        {
          red = 0.0 ;
          if( COLOR_MODE != SYCLIC_SCALE)
         {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
         else {
                        Color[Vertice_no][0]= 0.0;
                        Color[Vertice_no][1]= 0.0;
                        Color[Vertice_no][2]= 1.0;
                        Color[Vertice_no][3]= 1.0;
                }
         return 1 ;
        }

        if(Temperature[Vertice_no][0] >10 && Temperature[Vertice_no][0] <= 15)// Dirty_Green

        {
         red = 1.0/16.0 ;
         if( COLOR_MODE != SYCLIC_SCALE)
         {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
         else    {
                Color[Vertice_no][0]= 0.0625;
                Color[Vertice_no][1]= 0.3137;
                Color[Vertice_no][2]= 0.0;
                Color[Vertice_no][3]= 1.0;
                }
         return 1 ;
        }

        if(Temperature[Vertice_no][0] >15 && Temperature[Vertice_no][0] <= 20)//Light_Green

        {
                red = 2.0/16.0 ;
                if( COLOR_MODE != SYCLIC_SCALE)
         {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
        else    {
                Color[Vertice_no][0]= 0.1250;
```

```
            Color[Vertice_no][1]= 0.8784;
            Color[Vertice_no][2]= 0.0;
            Color[Vertice_no][3]= 1.0;
            }

 return 1 ;
 }

 if(Temperature[Vertice_no][0] >20 && Temperature[Vertice_no][0] <= 25)//Light_Blue

 {
  red = 3.0/16.0 ;
  if( COLOR_MODE != SYCLIC_SCALE)
  {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
  else     {
            Color[Vertice_no][0]= 0.1875;
            Color[Vertice_no][1]= 1.0;
            Color[Vertice_no][2]= 1.0;
            Color[Vertice_no][3]= 1.0;
            }

 return 1 ;
 }

 if(Temperature[Vertice_no][0] >25 && Temperature[Vertice_no][0] <= 30)//Brown

 {
  red = 4.0/16.0 ;
  if( COLOR_MODE != SYCLIC_SCALE)
  {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
else       {
            Color[Vertice_no][0]= 0.25;
            Color[Vertice_no][1]= 0.25;
            Color[Vertice_no][2]= 0.0;
            Color[Vertice_no][3]= 1.0;
            }

            return 1 ;
}

if(Temperature[Vertice_no][0] >30 && Temperature[Vertice_no][0] <= 35)// Purplish

{
 red = 5.0/16.0 ;
 if( COLOR_MODE != SYCLIC_SCALE)
 {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
else       {
            Color[Vertice_no][0]= 0.3125;
            Color[Vertice_no][1]= 0.0;
            Color[Vertice_no][2]= 1.0;
            Color[Vertice_no][3]= 1.0;
            }

return 1 ;
```

```
}

if(Temperature[Vertice_no][0] >35 && Temperature[Vertice_no][0] <= 40)  //Gray

{
 red = 6.0/16.0 ;
 if( COLOR_MODE != SYCLIC_SCALE)
 {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
 else {
        Color[Vertice_no][0]= 0.3750;
        Color[Vertice_no][1]= 0.3750;
        Color[Vertice_no][2]= 0.3750;
        Color[Vertice_no][3]= 1.0;
        }

return 1 ;
}

if(Temperature[Vertice_no][0] >40 && Temperature[Vertice_no][0] <= 45)//Dark_Brown

{
 red = 7.0/16.0 ;
 if( COLOR_MODE != SYCLIC_SCALE)
 {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
 else {
        Color[Vertice_no][0]= 0.4375;
        Color[Vertice_no][1]= 0.5843;
        Color[Vertice_no][2]= 0.0;
        Color[Vertice_no][3]= 1.0;
        }

return 1 ;
}

if(Temperature[Vertice_no][0] >45 && Temperature[Vertice_no][0] <= 50)  // Cherry

{
 red = 8.0/16.0 ;
 if( COLOR_MODE != SYCLIC_SCALE)
 {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
else    {
        Color[Vertice_no][0]= 0.5;
        Color[Vertice_no][1]= 0.0;
        Color[Vertice_no][2]= 0.5;
        Color[Vertice_no][3]= 1.0;
         }

return 1 ;
}

if(Temperature[Vertice_no][0] >50 && Temperature[Vertice_no][0] <= 55) //Fading_Red

{
 red = 9.0/16.0 ;
```

```
   if( COLOR_MODE != SYCLIC_SCALE)
   {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
   else      {
             Color[Vertice_no][0]= 0.5625;
             Color[Vertice_no][1]= 0.1568;
             Color[Vertice_no][2]= 0.5607;
             Color[Vertice_no][3]= 1.0;
              }

  return 1 ;
 }

 if(Temperature[Vertice_no][0] >55 && Temperature[Vertice_no][0] <= 60)  //  Purple

 {
  red = 10.0/16.0 ;
  if( COLOR_MODE != SYCLIC_SCALE)
  {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
  else      {
            Color[Vertice_no][0]= 0.6250;
            Color[Vertice_no][1]= 0.0;
            Color[Vertice_no][2]= 1.0;
            Color[Vertice_no][3]= 1.0;
            }
 return 1 ;
 }
 if(Temperature[Vertice_no][0] >60 && Temperature[Vertice_no][0] <= 65)//Dark_Gold
 {
  red = 11.0/16.0 ;
  if( COLOR_MODE != SYCLIC_SCALE)
  {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
 else      {
           Color[Vertice_no][0]= 0.6875;
           Color[Vertice_no][1]= 0.8156;
           Color[Vertice_no][2]= 0.0;
           Color[Vertice_no][3]= 1.0;
           }

 return 1 ;
 }

 if(Temperature[Vertice_no][0] >65 && Temperature[Vertice_no][0] <= 70)//Dirty_Red

 {
  red = 12.0/16.0 ;
  if( COLOR_MODE != SYCLIC_SCALE)
  {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
  else      {
           Color[Vertice_no][0]= 0.75;
           Color[Vertice_no][1]= 0.0;
           Color[Vertice_no][2]= 0.3960;
           Color[Vertice_no][3]= 1.0;
           }
```

```
return 1 ;
}

if(Temperature[Vertice_no][0] >70 && Temperature[Vertice_no][0] <= 75)//Dirty_Pink

{
 red = 13.0/16.0 ;
 if( COLOR_MODE != SYCLIC_SCALE)
 {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
else    {
                Color[Vertice_no][0]= 0.8125;
                Color[Vertice_no][1]= 0.3725;
                Color[Vertice_no][2]= 1.0;
                Color[Vertice_no][3]= 1.0;
        }

return 1 ;
}
if(Temperature[Vertice_no][0] >75 && Temperature[Vertice_no][0] <= 80)//Pinkish

{
 red = 14.0/16.0 ;
 if( COLOR_MODE != SYCLIC_SCALE)
 {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
 else   {
                Color[Vertice_no][0]= 0.8750;
                Color[Vertice_no][1]= 0.0;
                Color[Vertice_no][2]= 1.0;
                Color[Vertice_no][3]= 1.0;
        }

return 1 ;
}

if(Temperature[Vertice_no][0] >80 && Temperature[Vertice_no][0] <= 85)  //  Gold

{
 red = 15.0/16.0 ;
 if( COLOR_MODE != SYCLIC_SCALE)
 {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
 else   {
                Color[Vertice_no][0]= 0.9375;
                Color[Vertice_no][1]= 0.9490;
                Color[Vertice_no][2]= 0.0;
                Color[Vertice_no][3]= 1.0;
        }

return 1 ;
}
if(Temperature[Vertice_no][0] >85 )// 85<T Degrees Dark_Red
{
 red = 16.0/16.0 ;
 if( COLOR_MODE != SYCLIC_SCALE)
 {Modify_Color_Array_With_A_Given_Mode(Vertice_no,COLOR_MODE, red);}
```

```
        else {
                        Color[Vertice_no][0]= 1.0;
                        Color[Vertice_no][1]= 0.0;
                        Color[Vertice_no][2]= 0.0;
                        Color[Vertice_no][3]= 1.0;
                }
        return 1 ;
        }
        return 1;
}// End of function Fill_Up_Color_Table
/*******************************************************************************
* This function reads the display screen pixels from the frame buffer with a 100 by 100      *
*  square and finds the avarage red value of square and finally after reading the last square     *
* of the screen it finds the  maximum red value of the screen .                                   *
*******************************************************************************/
void Read_Screen()
{

float sum_of_red = 0.0,avr_red = 0.0;

int index = 0 , deb = 0;


 for (int y = 900 ;y>=0;y=y-100)
 {
   for(int x = 0;x <=1000 ; x= x + 100)
   {
        glReadPixels (x,y,100,100,GL_RED,GL_UNSIGNED_BYTE,buffer1);

        for (int counter = 0;counter < 10000 ;counter++)
        {
                // We defined the buffer1 array as character to make it occupy less memory space.
                sum_of_red=sum_of_red+int(buffer1[counter]);

        }//End of for(counter)
        avr_red=sum_of_red/10000;
        avarage_red_datas[index][0]=avr_red ;
        avarage_red_datas[index][1]=x;
        avarage_red_datas[index][2]=y;
        index=index+1 ;

        sum_of_red = 0.0 ;
   }//End of for(x)..


 } // End of for(y)..


}// End of Read Buffer...
```

```
/*****************************************************************************
* This function reads the display screen pixels from the frame buffer with a 100 by 100     *
*  square and finds the maximum and minimum illuminated pixel in that square and finds      *
* a contrast value for that square by subtructing the max from min illumination value      *
* After finding the contrast values for all squares it picks up the max. contrast of the   *
* screen                                                                                    *
*****************************************************************************/

void Read_TV_Screen()
{

int       max_contrast = 0,min_contrast = 255;
int       max_contrast_x ,max_contrast_y,
          min_contrast_x ,min_contrast_y ;
float      Contrast[110];


int index = 0;


 for (int y = 800 ;y>= 100;y=y-100)
 {
   for(int x =100 ;x <= 900 ; x= x + 100)
   {
        glReadPixels (x,y,100,100,GL_LUMINANCE,GL_UNSIGNED_BYTE,buffer2);

        for (int counter = 0;counter < 10000 ;counter++)
        {
           if(int(buffer2[counter]) > max_contrast)
           {
                max_contrast = int(buffer2[counter]);
                max_contrast_x = x;
                max_contrast_y = y;
           }

        if(int(buffer2[counter]) < min_contrast)
           {
                min_contrast = int(buffer2[counter]);
                min_contrast_x = x;
                min_contrast_y = y;
           }

        }//End of for(counter)

        Contrast[index] = max_contrast- min_contrast ;
        if (Contrast[index] > Max_Contrast_Of_Scene){
                Max_Contrast_Of_Scene = Contrast[index] ;
                Max_Contrast_X = max_contrast_x ;
                Max_Contrast_Y = max_contrast_y ;
                }//End of if(...).

        // Initialize the values for comparison purposes. If they are not initialized we never find
        // the correct min and max values.
        max_contrast = 0;
```

```
            min_contrast = 255 ;

            index=index+1 ;
     }  //End of for(x)..
     } // End of for(y)..

}// End of Read_TV_Screen....


/*****************************************************************************
* This is a convenience routine for finding the index value of global "avarage_red_datas"    *
* array which has the max red value in it.                                                   *
*****************************************************************************/
int  Find_Index_For_Max_Value()
{
          int index ;

          for(int i = 0; i < 110 ;i++)
          {
             if(avarage_red_datas[i][0] > MAX_RED)
             {
                   MAX_RED = avarage_red_datas[i][0];
                   index = i;
             }

          }

          return index ;

}//End of Find_Max_Value.....
```

```
/***************************************************************************
* This function prints LOCK_ON to the screen with big fonts.              *
***************************************************************************/

void Print_Lock_On (int xpos , int ypos)
{

// Since we are going to write on a 2D screen we must use the Orthographic projection.
  glMatrixMode(GL_PROJECTION);
  glPushMatrix() ;
  glLoadIdentity();
  gluOrtho2D(0.0,1400.0,0.0,800.0);


  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glLoadIdentity();
  glPushAttrib(GL_COLOR_BUFFER_BIT);
  glColor3f(0.1, 1.0, 0.5);

 // Do a setpoint for the lower lefthand coordinate of the text string.
  glRasterPos3f(xpos,  ypos,  0.0);
  drawCharstring(fontHandle1, "LOCKED ON!");
  glPopAttrib();
  glMatrixMode(GL_MODELVIEW);
  glPopMatrix();
  glMatrixMode(GL_PROJECTION);
  glPopMatrix();
}


/***************************************************************************
* This function prints the temperature equilant values of the colors on each color box on    *
* bottom of the screen.                                                                       *
***************************************************************************/
void Print_Numbers_On_Color_Scale()
{
  glMatrixMode(GL_PROJECTION);
  glPushMatrix() ;
  glLoadIdentity();
  gluOrtho2D(0.0,1400.0,0.0,800.0);


  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glLoadIdentity();
  glColor3f(1.0, 1.0, 1.0);

  glRasterPos3f(175.0,63.0,0.0);
  drawCharstring(fontHandle2, "10");
  glRasterPos3f(225.0,63.0,0.0);
  drawCharstring(fontHandle2, "15");
  glRasterPos3f(275.0,63.0,0.0);
  drawCharstring(fontHandle2, "20");
```

```
    glRasterPos3f(325.0,63.0,0.0);
    drawCharstring(fontHandle2, "25");
    glRasterPos3f(375.0,63.0,0.0);
    drawCharstring(fontHandle2, "30");
    glRasterPos3f(425.0,63.0,0.0);
    drawCharstring(fontHandle2, "35");
    glRasterPos3f(475.0,63.0,0.0);
    drawCharstring(fontHandle2, "40");
    glRasterPos3f(525.0,63.0,0.0);
    drawCharstring(fontHandle2, "45");
    glRasterPos3f(575.0,63.0,0.0);
    drawCharstring(fontHandle2, "50");
    glRasterPos3f(625.0,63.0,0.0);
    drawCharstring(fontHandle2, "55");
    glRasterPos3f(675.0,63.0,0.0);
    drawCharstring(fontHandle2, "60");
    glRasterPos3f(725.0,63.0,0.0);
    drawCharstring(fontHandle2, "65");
    glRasterPos3f(775.0,63.0,0.0);
    drawCharstring(fontHandle2, "70");
    glRasterPos3f(825.0,63.0,0.0);
    drawCharstring(fontHandle2, "75");
    glRasterPos3f(875.0,63.0,0.0);
    drawCharstring(fontHandle2, "80");
    glRasterPos3f(925.0,63.0,0.0);
    drawCharstring(fontHandle2, "85");
    glRasterPos3f(975.0,63.0,0.0);
    drawCharstring(fontHandle2, "90");


    glMatrixMode(GL_MODELVIEW);
    glPopMatrix();
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();

}
```

```
/****************************************************************************
* This function finds the total radiance for a given temperature.           *
****************************************************************************/
float Find_Total_Radiance(float Temp)
{
        float     total_radiance = 0.0,e = 1.0 ;

        total_radiance=e *sigma*Temp*Temp*Temp*Temp;
        return total_radiance ;
}


/****************************************************************************
* This function apply the atmospheric extinction to the signal and finds the equilant   *
* temp.  value at the sensor .                                               *
****************************************************************************/
float Temp_At_Sensor(float total_radiance , float mu )
{
        float pas, Temp ,R = (Firing_Distance/100),e = 1.0 ;

        pas=(total_radiance/PI)*exp(-mu*R);
        Temp=sqrt(sqrt(pas/(e*sigma)));
        return Temp;
}


/****************************************************************************
* This function modifies the position of the box that we draw to simulate the scanning  *
* function of the sensor.                                                              *
****************************************************************************/
void Update_ROW_COLOMN(int Row, int Colomn)
{
if(Row > 1000)
   {
        ROW = 0;
        COLOMN = COLOMN - 100 ;
        if(Colomn <= 0)
        {
          COLOMN = 900;
          SEARCHED_ONES = 1;
        }// End of if(Colomn)......
   }//End of if(Row)......
   else ROW = ROW + 100 ;
}// End of Update_ROW_COLOMN(..)
```

```
/***************************************************************************
* This function is called from "draw_the_scene" function if the sensor is IR type.       *
***************************************************************************/

 void Call_IR_Functions()
{


  if(!SEARCHED_ONES)  // Then the box we are drawing to simulate the scanning doesn't
                                        // completly scanned the whole display screen.So go ahead and
                                       // update the position of the box for the next frame.
  {

    draw_a_rectangle(ROW,COLOMN);
    Update_ROW_COLOMN(ROW,COLOMN);

  }// End of if(!SEARCHED_ONES).....

  else    // Yes we completed scanning the sceene..
  {
    if(!READ_ONES)    //  Then we haven't read the screen previously.
    {
          Read_Screen ();
          INDEX_FOR_MAX_RED = Find_Index_For_Max_Value( ) ;
          READ_ONES = 1;
    }
    else   // It means this screen reading is at least second reading.
    {
          if(MAX_RED > 55.0)   //  55 is our treshhold value for red pixels.It is approximately
                                              //  equial to 0.2 red value of the pixel where 1.0 being the max.
          {
            if(MAX_X == OLD_MAX_X && MAX_Y == OLD_MAX_Y)
            {
             Print_Lock_On (400,150);

            }
            if(MAX_X <= 1000 && MAX_Y < 1000) // We are inside the display screen.
            {
                   draw_a_rectangle(int(avarage_red_datas[INDEX_FOR_MAX_RED][1]),
                               int(avarage_red_datas[INDEX_FOR_MAX_RED][2])+50);
            }


       }// End of if(MAX_RED > 55.0)..........

     }//End of else....

   }//End of else......

// Here we draw the color scale in two dimensional screen...
 if(MODE == SYCLIC_SCALE)  {DrawTheSyclicColorScale();}
 else                      {DrawTheColorScale(MODE);  }

  Print_Numbers_On_Color_Scale();
}//End of Call_IR_Functions()



/*************************************************************************
* This function is called from "draw_the_scene" function if the sensor is IR type.          *
*************************************************************************/
 void Call_TV_Functions()
{


 if(!SEARCHED_ONES)  // Then the box we are drawing to simulate the scanning doesn't
                                      // completly scanned the whole display screen.So go ahead and
                                     // update the position of the box for the next frame.
 {

   draw_a_rectangle(ROW,COLOMN);
   Update_ROW_COLOMN(ROW,COLOMN);

 }// End of if(!SEARCHED_ONES).....

 else    // Yes we completed scanning the sceene..
 {
   if(!READ_ONES)    //  Then we haven't read the screen previously.
   {

         Read_TV_Screen ();
         if(Max_Contrast_Of_Scene > 55.0)
         {
            draw_a_rectangle(Max_Contrast_X,Max_Contrast_Y + 50);

         }// End of Max_Contrast_Of_Scene > 55.0).


         READ_ONES = 1;
   }
   else
   {
         if(Max_Contrast_Of_Scene > 55.0)
         {
            Print_Lock_On (400,150);
            draw_a_rectangle(Max_Contrast_X,Max_Contrast_Y + 50);

         }// End of Max_Contrast_Of_Scene > 55.0).

    }//End of else....

   }//End of else......

}//End of Call_IR_Functions()


/*************************************************************************
```

```
* This function  modifies the color array with respect to the color mode sellected.        *
**************************************************************************/
void Modify_Color_Array_With_A_Given_Mode(int Vertice_no, int SELECTED_MODE, double red_value)
{

        switch(SELECTED_MODE)
        {

                case GRAY_SCALE:
                Color[Vertice_no][0]= red_value ;
                Color[Vertice_no][1]= red_value ;
                Color[Vertice_no][2]= red_value ;
                Color[Vertice_no][3]= 1.0;
                break;

                case TEMPERATURE_SCALE:
                Color[Vertice_no][0]= red_value ;
                Color[Vertice_no][1]= 0.0;
                Color[Vertice_no][2]= (1-red_value);
                Color[Vertice_no][3]= 1.0;
                break;


                default:
                break;

        }//End of switch..
}//End of Modify_Color_Table_With.....
```

```
/*************************************************************************************
*  This is materialsupport.C                                                         *
*  Main body of this  ".C" file is taken from CS4202 course notes and some material  *
*  definitions are added. For example originally in this file "emission type" was not *
*  included in definitions of the properties of the materials. Since sun has it's own emission *
* We seperatly defined a function call for "SUN" material.A function to enable and disable *
* the fog is also defined for convience.                                             *
*************************************************************************************/
// These functions are self-contained and can be used by you in your applications.


#include <Xm/Xm.h>          // Get the Motif stuff...

#include <GL/GLwMDrawA.h>    // We are going to use an OpenGL Motif Draw widget

#include <GL/gl.h>          // Get the OpenGL required includes.
#include <GL/glu.h>
#include <GL/glx.h>

#include <iostream.h>        // C++ I/O subsystem.
#include <math.h>
#include <stdlib.h>         // Get exit() function.

#include "materialsupport.h"  // Get the names of the available materials.

#include "materialsupport_funcs.h"  // Get material support function names.


/*************************************************************************
*  The following function turns on the Light Model.
*************************************************************************/
void turnOnTheLightModel()
{

  // A dim white for the lighting model ambient color.
  GLfloat lmodel_ambient[] = { 0.2, 0.2, 0.2, 1.0 };


  // Set the global ambient light intensity.
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);

  // Set whether the viewpoint position is local to the scene or
  // whether it should be considered to be an infinite distance away.
  // We have selected an infinite viewpoint.
  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE);

  // Say that we want two-sided lighting.
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

  // Enable the lighting model (there is a glDisable()).
  glEnable(GL_LIGHTING);

}
```

```
/****************************************************************************
 *  The following function turns off the Light Model.
****************************************************************************/
void turnOffTheLightModel()
{

  glDisable(GL_LIGHTING);

}

/****************************************************************************
 *  The following function turns on the Sun Light
****************************************************************************/

void turnOnTheLights()
{

  GLfloat light_ambient[] = { 0.5, 0.5, 0.5, 1.0 };

  GLfloat light_diffuse[] = { 0.7, 0.7, 0.7, 1.0 };

  GLfloat light_diffuse1[] = { 0.7, 0.7, 0.7, 1.0 };

  GLfloat light_specular[] = { 0.5,0.5, 0.5, 1.0 };

  GLfloat light_specular1[] = { 1.0, 1.0, 1.0, 1.0 };

  // The ending 0.0 means directional light at infinity.
  GLfloat light_position[] = { 0.0, 100.0, -100.0, 0.0 };

  GLfloat light2_position[] = { 0.0, 0.0, -100.0, 0.0 };

  // Specify the ambient rgba for the lights.
  glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);



  // Specify the diffuse rgba for the lights.
  glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);


  // Specify the specular rgba for the lights.
  glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);

// The position of the  light is moving.So we didn't specify here..
// glLightPosition(GL_LIGHT0,.............);


  // Turn the light on ...
  glEnable(GL_LIGHT0);

}
```

```
/***************************************************************************
*  The following function turns off the  Sun Light
***************************************************************************/

void turnOffTheLights()
{

  glDisable(GL_LIGHT0);

}

/***************************************************************************
*  The following function enables the fog.
***************************************************************************/

//Here we are enabling the fog so the objects in the far distance look blury..
void EnableFog(float density)
{

        //Apply Extinction to Electro-optical Signal
        GLint fogmode;
        glEnable(GL_FOG);
        {
                GLfloat fogcolor[4]={0.8,0.894,0.815,0.5};
                fogmode=GL_EXP;
                glFogi(GL_FOG_MODE,fogmode);
                glFogfv(GL_FOG_COLOR,fogcolor);
                glFogf(GL_FOG_DENSITY,density);
                glHint(GL_FOG_HINT,GL_NICEST);
                glClearColor(0.3921,0.3921,0.3921,0.5);
        }

}//End of EnableFog........

/***************************************************************************
*  The following function disables the fog
***************************************************************************/

//Disable the fog here..
void DisableFog()
{
        glDisable(GL_FOG);
}

/***************************************************************************
* Turn on a Material for a particular face. I(Prof. Zyda) collected a number of materials
* I found and made them easily accessible via define constant.
* Options for whichFace = GL_FRONT, GL_BACK & others...
***************************************************************************/
void turnOnMaterial(GLenum whichFace, int whichMaterial)
{

  // Set the appropriate material type...
```

```
switch(whichMaterial)
{

  case BRASS:
    {
      // Here are the brass material values ...
      GLfloat brass_ambient[] = { 0.35, 0.25, 0.1, 1.0 };
      GLfloat brass_diffuse[] = { 0.65, 0.5, 0.35, 1.0 };
      GLfloat brass_specular[] = { 0.65, 0.5, 0.35, 1.0 };
      GLfloat brass_shininess[] = { 5.0 };

      // Make the brass material calls.
      makeGLMaterialCalls(whichFace,
                          brass_ambient,
                          brass_diffuse,
                          brass_specular,
                          brass_shininess);
    }
    break;

  case SHINYBRASS:
    {
      // Here are the shinybrass material values ...
      GLfloat shinybrass_ambient[] = { 0.25, 0.15, 0.0, 1.0 };
      GLfloat shinybrass_diffuse[] = { 0.65, 0.5, 0.35, 1.0 };
      GLfloat shinybrass_specular[] = { 0.9, 0.6, 0.0, 1.0 };
      GLfloat shinybrass_shininess[] = { 10.0 };

      // Make the shinybrass material calls.
      makeGLMaterialCalls(whichFace,
                          shinybrass_ambient,
                          shinybrass_diffuse,
                          shinybrass_specular,
                          shinybrass_shininess);
    }
    break;

  case PEWTER:
    {
      // Here are the pewter material values ...
      GLfloat pewter_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
      GLfloat pewter_diffuse[] = { 0.6, 0.55, 0.65, 1.0 };
      GLfloat pewter_specular[] = { 0.9, 0.9, 0.95, 1.0 };
      GLfloat pewter_shininess[] = { 10.0 };

      // Make the pewter material calls.
      makeGLMaterialCalls(whichFace,
                          pewter_ambient,
                          pewter_diffuse,
                          pewter_specular,
                          pewter_shininess);
    }
    break;
```

```
case SILVER:
  {
    // Here are the silver material values ...
    GLfloat silver_ambient[] = { 0.4, 0.4, 0.4, 1.0 };
    GLfloat silver_diffuse[] = { 0.3, 0.3, 0.3, 1.0 };
    GLfloat silver_specular[] = { 0.9, 0.9, 0.95, 1.0 };
    GLfloat silver_shininess[] = { 30.0 };

    // Make the silver material calls.
    makeGLMaterialCalls(whichFace,
                        silver_ambient,
                        silver_diffuse,
                        silver_specular,
                        silver_shininess);
  }
  break;

case GOLD:
  {
    // Here are the gold material values ...
    GLfloat gold_ambient[] = { 0.4, 0.2, 0.0, 1.0 };
    GLfloat gold_diffuse[] = { 0.9, 0.5, 0.0, 1.0 };
    GLfloat gold_specular[] = { 0.7, 0.7, 0.0, 1.0 };
    GLfloat gold_shininess[] = { 10.0 };

    // Make the gold material calls.
    makeGLMaterialCalls(whichFace,
                        gold_ambient,
                        gold_diffuse,
                        gold_specular,
                        gold_shininess);
  }
  break;

case SHADOW:
  {
               // Here are the shadow values ...
      GLfloat Ambient[] = { 0.0, 0.0, 0.0, 0.4 };
      GLfloat Diffuse[] = { 0.0, 0.0, 0.0, 0.4 };
      GLfloat Specular[] = { 0.0, 0.0, 0.0, 0.4 };
      GLfloat Shininess[] = { 0.0 };


    // Make the shinygold material calls.
    makeGLMaterialCalls(whichFace,
                        Ambient,
                        Diffuse,
                        Specular,
                        Shininess);
  }
  break;

case PLASTER:
  {
      // Here are the plaster material values ...
      GLfloat plaster_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
      GLfloat plaster_diffuse[] = { 0.95, 0.95, 0.95, 1.0 };
      GLfloat plaster_specular[] = { 0.0, 0.0, 0.0, 1.0 };
      GLfloat plaster_shininess[] = { 1.0 };

      // Make the plaster material calls.
      makeGLMaterialCalls(whichFace,
                          plaster_ambient,
                          plaster_diffuse,
                          plaster_specular,
                          plaster_shininess);
    }
    break;

  case REDPLASTIC:
    {
      // Here are the redplastic material values ...
      GLfloat redplastic_ambient[] = { 0.3, 0.1, 0.1, 1.0 };
      GLfloat redplastic_diffuse[] = { 0.5, 0.1, 0.1, 1.0 };
      GLfloat redplastic_specular[] = { 0.45, 0.45, 0.45, 1.0 };
      GLfloat redplastic_shininess[] = { 30.0 };

      // Make the redplastic material calls.
      makeGLMaterialCalls(whichFace,
                          redplastic_ambient,
                          redplastic_diffuse,
                          redplastic_specular,
                          redplastic_shininess);
    }
    break;

  case GREENPLASTIC:
    {
      // Here are the greenplastic material values ...
      GLfloat greenplastic_ambient[] = { 0.1, 0.3, 0.1, 1.0 };
      GLfloat greenplastic_diffuse[] = { 0.1, 0.5, 0.1, 1.0 };
      GLfloat greenplastic_specular[] = { 0.45, 0.45, 0.45, 1.0 };
      GLfloat greenplastic_shininess[] = { 30.0 };

      // Make the greenplastic material calls.
      makeGLMaterialCalls(whichFace,
                          greenplastic_ambient,
                          greenplastic_diffuse,
                          greenplastic_specular,
                          greenplastic_shininess);
    }
    break;

  case BLUEPLASTIC:
    {
      // Here are the blueplastic material values ...
      GLfloat blueplastic_ambient[] = { 0.1, 0.1, 0.3, 1.0 };
      GLfloat blueplastic_diffuse[] = { 0.1, 0.1, 0.5, 1.0 };
```

```
          GLfloat blueplastic_specular[] = { 0.45, 0.45, 0.45, 1.0 };
          GLfloat blueplastic_shininess[] = { 30.0 };

          // Make the blueplastic material calls.
          makeGLMaterialCalls(whichFace,
                        blueplastic_ambient,
                        blueplastic_diffuse,
                        blueplastic_specular,
                        blueplastic_shininess);
        }
        break;

      case RED:
        {
          // Here are the red material values...
          GLfloat red_ambient[] = { 0.2, 0.0, 0.0, 1.0 };
          GLfloat red_diffuse[] = { 1.0, 0.0, 0.0, 1.0 };
          GLfloat red_specular[] = { 1.0, 0.0, 0.0, 1.0 };
          GLfloat red_shininess[] = { 1.0 };

          // Make the red material calls.
          makeGLMaterialCalls(whichFace,
                        red_ambient,
                        red_diffuse,
                        red_specular,
                        red_shininess);
        }
        break;

      case WHITE:
        {
          // Here are the white material values...
          GLfloat white_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
          GLfloat white_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
          GLfloat white_specular[] = { 1.0, 1.0, 1.0, 1.0 };
          GLfloat white_shininess[] = { 1.0 };

          // Make the red material calls.
          makeGLMaterialCalls(whichFace,
                        white_ambient,
                        white_diffuse,
                        white_specular,
                        white_shininess);
        }
        break;
    case t72mat0 :
    {
    GLfloat t72mat0_ambient[] = { 0.039216,0.054902,0.039216};
    GLfloat t72mat0_diffuse[] = { 0.196078,0.274510,0.196078};
    GLfloat t72mat0_specular[] = { 0.000000,0.000000,0.000000};
    GLfloat t72mat0_shininess[] = { 0.000000 };
     // Make the brass material calls.
    makeGLMaterialCalls(whichFace,t72mat0_ambient ,t72mat0_diffuse ,t72mat0_specular
    ,t72mat0_shininess );
```

```
}
break ;
case t72mat1 :
{
GLfloat t72mat1_ambient[] = { 0.039216,0.078431,0.039216};
GLfloat t72mat1_diffuse[] = { 0.196078,0.392157,0.196078};
GLfloat t72mat1_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72mat1_shininess[] = { 0.000000 };
// Make the brass material calls.
makeGLMaterialCalls(whichFace,t72mat1_ambient ,t72mat1_diffuse ,t72mat1_specular
,t72mat1_shininess );
}
break ;
case t72mat2 :
{
GLfloat t72mat2_ambient[] = { 0.039216,0.039216,0.039216};
GLfloat t72mat2_diffuse[] = { 0.196078,0.196078,0.196078};
GLfloat t72mat2_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72mat2_shininess[] = { 0.000000 };
// Make the brass material calls.
makeGLMaterialCalls(whichFace,t72mat2_ambient ,t72mat2_diffuse ,t72mat2_specular
,t72mat2_shininess );
}
break ;
case t72mat3 :
{
GLfloat t72mat3_ambient[] = { 0.039216,0.070588,0.039216};
GLfloat t72mat3_diffuse[] = { 0.196078,0.352941,0.196078};
GLfloat t72mat3_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72mat3_shininess[] = { 0.000000 };
// Make the brass material calls.
makeGLMaterialCalls(whichFace,t72mat3_ambient ,t72mat3_diffuse ,t72mat3_specular
,t72mat3_shininess );
}
break ;
case t72mat4 :
{
GLfloat t72mat4_ambient[] = { 0.039216,0.062745,0.039216};
GLfloat t72mat4_diffuse[] = { 0.196078,0.313725,0.196078};
GLfloat t72mat4_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72mat4_shininess[] = { 0.000000 };
// Make the brass material calls.
makeGLMaterialCalls(whichFace,t72mat4_ambient ,t72mat4_diffuse ,t72mat4_specular
,t72mat4_shininess );
}
break ;
case t72mat5 :
{
GLfloat t72mat5_ambient[] = { 0.047059,0.047059,0.047059};
GLfloat t72mat5_diffuse[] = { 0.235294,0.235294,0.235294};
GLfloat t72mat5_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72mat5_shininess[] = { 0.000000 };
// Make the brass material calls.
makeGLMaterialCalls(whichFace,t72mat5_ambient ,t72mat5_diffuse ,t72mat5_specular
,t72mat5_shininess );
}
break ;
case t72mat6 :
{
GLfloat t72mat6_ambient[] = { 0.039216,0.094118,0.039216};
GLfloat t72mat6_diffuse[] = { 0.196078,0.470588,0.196078};
GLfloat t72mat6_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72mat6_shininess[] = { 0.000000 };
 // Make the brass material calls.
makeGLMaterialCalls(whichFace,t72mat6_ambient ,t72mat6_diffuse ,t72mat6_specular
,t72mat6_shininess );
}
break ;
case t72mat7 :
{
GLfloat t72mat7_ambient[] = { 0.039216,0.086275,0.039216};
GLfloat t72mat7_diffuse[] = { 0.196078,0.431373,0.196078};
GLfloat t72mat7_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72mat7_shininess[] = { 0.000000 };
 // Make the brass material calls.
makeGLMaterialCalls(whichFace,t72mat7_ambient ,t72mat7_diffuse ,t72mat7_specular
,t72mat7_shininess );
}
break ;
case t72mat8 :
{
GLfloat t72mat8_ambient[] = { 0.062745,0.062745,0.062745};
GLfloat t72mat8_diffuse[] = { 0.313725,0.313725,0.313725};
GLfloat t72mat8_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72mat8_shininess[] = { 0.000000 };
 // Make the brass material calls.
makeGLMaterialCalls(whichFace,t72mat8_ambient ,t72mat8_diffuse ,t72mat8_specular
,t72mat8_shininess );
}
break ;
case t72mat9 :
{
GLfloat t72mat9_ambient[] = { 0.039216,0.047059,0.039216};
GLfloat t72mat9_diffuse[] = { 0.196078,0.235294,0.196078};
GLfloat t72mat9_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72mat9_shininess[] = { 0.000000 };
 // Make the brass material calls.
makeGLMaterialCalls(whichFace,t72mat9_ambient ,t72mat9_diffuse ,t72mat9_specular
,t72mat9_shininess );
}
break ;
case t72_3mat0 :
{
GLfloat t72_3mat0_ambient[] = { 0.039216,0.078431,0.039216};
GLfloat t72_3mat0_diffuse[] = { 0.196078,0.392157,0.196078};
GLfloat t72_3mat0_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72_3mat0_shininess[] = { 0.000000 };
 // Make the brass material calls.
```

```
makeGLMaterialCalls(whichFace,t72_3mat0_ambient ,t72_3mat0_diffuse ,t72_3mat0_specular
,t72_3mat0_shininess );
}
break ;
case t72_3mat1 :
{
GLfloat t72_3mat1_ambient[] = { 0.039216,0.094118,0.039216};
GLfloat t72_3mat1_diffuse[] = { 0.196078,0.470588,0.196078};
GLfloat t72_3mat1_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72_3mat1_shininess[] = { 0.000000 };
// Make the brass material calls.
makeGLMaterialCalls(whichFace,t72_3mat1_ambient ,t72_3mat1_diffuse ,t72_3mat1_specular
,t72_3mat1_shininess );
}
break ;
case t72_3mat2 :
{
GLfloat t72_3mat2_ambient[] = { 0.039216,0.062745,0.039216};
GLfloat t72_3mat2_diffuse[] = { 0.196078,0.313725,0.196078};
GLfloat t72_3mat2_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72_3mat2_shininess[] = { 0.000000 };
// Make the brass material calls.
makeGLMaterialCalls(whichFace,t72_3mat2_ambient ,t72_3mat2_diffuse ,t72_3mat2_specular
,t72_3mat2_shininess );
}
break ;
case t72_3mat3 :
{
GLfloat t72_3mat3_ambient[] = { 0.039216,0.070588,0.039216};
GLfloat t72_3mat3_diffuse[] = { 0.196078,0.352941,0.196078};
GLfloat t72_3mat3_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72_3mat3_shininess[] = { 0.000000 };
// Make the brass material calls.
makeGLMaterialCalls(whichFace,t72_3mat3_ambient ,t72_3mat3_diffuse ,t72_3mat3_specular
,t72_3mat3_shininess );
}
break ;
case t72_3mat4 :
{
GLfloat t72_3mat4_ambient[] = { 0.039216,0.039216,0.039216};
GLfloat t72_3mat4_diffuse[] = { 0.196078,0.196078,0.196078};
GLfloat t72_3mat4_specular[] = { 0.000000,0.000000,0.000000};
GLfloat t72_3mat4_shininess[] = { 0.000000 };
// Make the brass material calls.
makeGLMaterialCalls(whichFace,t72_3mat4_ambient ,t72_3mat4_diffuse ,t72_3mat4_specular
,t72_3mat4_shininess );
}
break ;

case SUN:
      {
        // Here are the pewter material values ...
        GLfloat SUN_ambient[] = { 1.0, 1.0, 1.0, 1.0 };
        GLfloat SUN_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
        GLfloat SUN_specular[] = { 1.0, 1.0, 1.0, 1.0 };
        GLfloat SUN_shininess[] = { 500.0 };
            GLfloat SUN_emission[] = { 1.0, 1.0, 1.0, 1.0 };

        // Make the pewter material calls.
        makeGLSunMaterialCalls(whichFace,
                        SUN_ambient,
                        SUN_diffuse,
                        SUN_specular,
                        SUN_shininess,
                                    SUN_emission );
      }
      break;

    default:
      break;

  }  // end switch statement.

}


// Here is the function called from turnOnMaterial
void makeGLMaterialCalls(GLenum whichFace,
                GLfloat *ambient,
                GLfloat *diffuse,
                GLfloat *specular,
                GLfloat *shininess)
{

  // Make the material calls.
  glMaterialfv(whichFace, GL_AMBIENT, ambient);
  glMaterialfv(whichFace, GL_DIFFUSE, diffuse);
  glMaterialfv(whichFace, GL_SPECULAR, specular);
  glMaterialfv(whichFace, GL_SHININESS, shininess);

}

// Here is the function that makes the actual gl calls for sun material.
void makeGLSunMaterialCalls(GLenum whichFace,
                GLfloat *ambient,
                GLfloat *diffuse,
                GLfloat *specular,
                GLfloat *shininess,
                         GLfloat *emission)
{

  // Make the material calls.
  glMaterialfv(whichFace, GL_AMBIENT, ambient);
  glMaterialfv(whichFace, GL_DIFFUSE, diffuse);
  glMaterialfv(whichFace, GL_SPECULAR, specular);
  glMaterialfv(whichFace, GL_SHININESS, shininess);
  glMaterialfv(whichFace, GL_EMISSION, emission);
}
```

```
/************************************************************************
* This is makeRasterFont.C                                              *
*                                                                       *
* This file contains two functions to support the reading of X fonts   *
* as raster bitmaps for use with OpenGL character display              *
************************************************************************/
#include <Xm/Xm.h>         // Get the Motif stuff...

#include <GL/GLwMDrawA.h>  // We are going to use an OpenGL Motif Draw widget

#include <X11/StringDefs.h>
#include <X11/keysym.h>

#include <GL/gl.h>     // Get the OpenGL required includes.
#include <GL/glu.h>
#include <GL/glx.h>

#include <iostream.h>  // Get the C++ I/O functions.

#include <stdlib.h>    // Get the exit() function.


/************************************************************************
* makeRasterFont                                                        *
*  -- make a set of display lists for a particular X font.             *
*  -- returns base of lists for use in drawing fonts.                  *
************************************************************************/
GLuint makeRasterFont(Widget w, char* fontname)
{

  XFontStruct *fontInfo;          // ptr to an X font.
  Font id;                        // Font id.
  unsigned int first, last;  // first and last char of the font.
  GLuint base;              // The base of the display lists.

  // See if we can load the X font...
  fontInfo = XLoadQueryFont(XtDisplay(w), fontname,);

  // Did we get a font?
  if(fontInfo == NULL)
  {
    cerr << "makeRasterFont: font not found = " << fontname << endl;
    exit(0);
  }

  // Get the font id.
  id = fontInfo->fid;

  // Get some ptr info for the font.
  first = fontInfo->min_char_or_byte2;
  last = fontInfo->max_char_or_byte2;

  // Compute the display lists for the font.
  base = glGenLists(last+1);
  if(base == 0)
  {
    cerr << "makeRasterFont: out of display list. " << endl;
    exit(0);
  }

  // Create bitmap display lists from an X font.
  glXUseXFont(id, first, last-first+1, base+first);

  // Return base
  return base;

} // end of makeRasterFont

/******************************************************************************
*                                                                             *
* drawCharstring(GLuint base, char *s)                                        *
*  -- draw the char string with the specified set of display lists.           *
*    -- base is the value returned from makeRasterFont, essentially  the base of a set   *
*  of display lists.                                                          *
******************************************************************************/
void drawCharstring(GLuint base, char *s)
{

  // Save the List Base Setting.
  glPushAttrib(GL_LIST_BIT);

  // Specify the display list base for CallLists.
  glListBase(base);

  // Draw the chars ...
  glCallLists(strlen(s), GL_UNSIGNED_BYTE, (unsigned char *)s);

  // Restore the List Base setting.
  glPopAttrib();

}
```

```
/*************************************************************************
 * This is file NPSimage.C                                               *
 * This file contains the methods that implement the  NPSimage class.    *
 *************************************************************************/

#include "NPSimage.h"

#include <string.h>

#include <iostream.h>     // Get the I/O stuff.

#include <GL/gl.h>        // Get the SGI routines.
#include <GL/glu.h>

#include "image.h"        // SGI image structures.


// External def for SGI image routines.

extern "C" IMAGE *iopen(const char *filename, char *access, ...);

extern "C" void getrow(IMAGE *image, unsigned short *buf, int y, int z);

extern "C" void putrow(IMAGE *image, unsigned short *buf, int y, int z);

extern "C" void iclose(IMAGE *);


// temp arrays to hold scratch info for processing an sgi image.
// RGBA order for the indices.
unsigned short buf[4][4096];




// Here is a constructor with no specs.
NPSimage::NPSimage()
{

  // If an NPSimage is previously defined, delete it.
 // NPSimage::~NPSimage();

  // We haven't been given an image size.

  // Create a default image.
  size[0] = 8;
  size[1] = 8;
  size[2] = 4;

  // Pt the image data at a simple 8 x 8 x 4 image.
  imagedata = new unsigned int [size[0] * size[1]];

  // No name for the image yet.
  name = new char [8];
  strcpy(name,"default");

}



// Here is the destructor for an NPSimage.
NPSimage::~NPSimage()
{

  // Call the del() function.
  del();

}



// Delete the image.
void NPSimage::del()
{

  // Delete the image data if it exists.
  if(imagedata != 0)
  {
    delete imagedata;
  }

  // Delete the image's name if it exists.
  if(name != 0)
  {
    delete name;
  }

}





// Create an empty image of a particular size.
// We pass in the name and size of the image to create.
NPSimage::NPSimage(char *name_img, int *size_img)
{

  // If an NPSimage is previously defined, delete it.
  NPSimage::~NPSimage();

  // Create an image.

  // Copy the size.
  size[0] = size_img[0];
  size[1] = size_img[1];
  size[2] = size_img[2];
```

```
    // Allocate an image of the appropriate size.
    imagedata = new unsigned int [size[0] * size[1]];

    // No name for the image yet.
    name = new char [strlen(name_img) + 1];
    strcpy(name, name_img);

  }



// Create an empty image of a particular size.
// We pass in the name and size of the image to create.
NPSimage::NPSimage(char *name_img, int xsize, int ysize, int zsize)
{

    // If an NPSimage is previously defined, delete it.
    NPSimage::~NPSimage();

    // Create an image.

    // Copy the size.
    size[0] = xsize;
    size[1] = ysize;
    size[2] = zsize;

    // Allocate an image of the appropriate size.
    imagedata = new unsigned int [size[0] * size[1]];

    // No name for the image yet.
    name = new char [strlen(name_img) + 1];
    strcpy(name, name_img);

}



// We need to read in an image from a particular file.
void NPSimage::read_from(const char *filename)
{

   IMAGE *image;           // a ptr to an SGI image structure.

   int x,y,z;              // Loop temps.

   unsigned int *ptr;      // Ptr to the image longs.


   // open an sgi image
   image = iopen(filename,"r");
   if(image == NULL)
   {
```

```
  cerr << "NPSimage::read_from: can't open input file " << filename << endl;
  return;
}

// If an NPSimage is previously defined, delete it.
NPSimage::~NPSimage();

// We have the image open and the old image deleted...

// Create an empty image of the right size.

// Copy the size.
size[0] = image->xsize;
size[1] = image->ysize;
size[2] = image->zsize;

// Allocate an image of the appropriate size.
imagedata = new unsigned int [size[0] * size[1]];

// Copy the filename into the image.
name = new char [strlen(filename) + 1];
strcpy(name, filename);


// get a pointer to the NPSimage longs.
ptr = imagedata;

// for each row of the image.
for(y=0; y < size[1]; y=y+1)
{

  for(z=0; z < size[2]; z=z+1)
  {

    // Read each row in reds, greens, blues, alpha order.
    getrow(image, buf[z], y, z);

  }

  //  we must now step across the row and set each long integer
  //  of the NPSimage format by combining the info from the sgi
  //  rows.
  for(x=0; x < size[0]; x=x+1)
  {

   // Someday we should generalize the below code to make it useable
   // for images with only 1 and 2 channels.

   //  compute the RGBa long to plug in and plug it in.
   if(size[2] == 4)
   {
    // Data reversed for OpenGL implementation.
    *ptr = (buf[0][x] << 24) + (buf[1][x] << 16) + (buf[2][x] << 8) + (buf[3][x]);
   }
```

```
        else
        {
         // RGBA image with alpha forced to 0xff
         // Originally: *ptr = buf[0][x] + (buf[1][x] << 8) + (buf[2][x] << 16) + (0xff << 24);
         // Images are stored in memory differently in OpenGL (RGBA) instead of
         // ABGR as in IRISGL.
         *ptr = (buf[0][x] << 24) + (buf[1][x] << 16) + (buf[2][x] << 8) + (0xff);
        }

        // step the ptr to the next long.
        ptr++;

      }

    }

   // Close the image file.
   iclose(image);

}



// We need to write out the image in SGI format.
void NPSimage::write_to(const char *filename)
{

  register IMAGE *image;   // a ptr to an SGI image structure.

  int x,y,z;               // Loop temps.

  unsigned int *ptr;       // Ptr to the image longs.


  // open an sgi rgb image for writing.
  image=iopen(filename,"w",RLE(1),size[2],size[0],size[1],size[2]);

  // get a pointer to the NPSimage longs.
  ptr = imagedata;

  //  for each row of the image ...
  for(y=0; y < size[1]; y=y+1)
  {

    // we must now step across the row and decode each integer
    // of the NPSimage format into the 16 bit shorts sgi requires.
    for(x=0; x < size[0]; x=x+1)
    {

      // Get the colors from the longs. Reversed for OpenGL.
      buf[0][x] = (unsigned short)((*ptr & 0xff000000) >> 24);
      buf[1][x] = (unsigned short)((*ptr & 0x00ff0000) >> 16);
      buf[2][x] = (unsigned short)((*ptr & 0x0000ff00) >> 8);
```

```
          buf[3][x] = (unsigned short)(*ptr & 0x000000ff);

          // step the ptr to the next long.
          ptr++;

        }

        // Write out the rows of the image.
        for(z=0; z < size[2]; z=z+1)
        {

          // Write each row in reds, greens, blues, alpha order.
          putrow(image, buf[z], y, z);

        }

      }

      // we must close the output sgi image file.
      iclose(image);


}




// We need to display the image.
void NPSimage::display()
{

  // Here is the OpenGL call to send the image to the open window.
  glDrawPixels(size[0], size[1], GL_RGBA, GL_UNSIGNED_BYTE, imagedata);

}




// The purpose of the = operator is for copying an NPSimage.
NPSimage&
NPSimage::operator=(NPSimage &img)
{

  int i;  // Loop temp


  // Delete the old image, if any.
  NPSimage::~NPSimage();

  // Copy the size.
  size[0] = img.size[0];
  size[1] = img.size[1];
  size[2] = img.size[2];

  // Allocate an image of the appropriate size.
  imagedata = new unsigned int [size[0] * size[1]];

  // Copy the image data from img to "this".
  for(i=0; i < (size[0] * size[1]); i=i+1)
  {
    imagedata[i] = img.imagedata[i];
  }

  // Copy the filename into the image.
  name = new char [strlen(img.name) + 1];
  strcpy(name, img.name);

  // Return a ptr to the new image.
  return *this;

}



// The purpose of the is_equal function is for comparing 2 NPSimages.
// We return 1 if the images are equal, otherwise 0.
int NPSimage::is_equal(NPSimage &img)
{

  int i;  // Loop temp


  // Compare the sizes.
  if(size[0] != img.size[0] || size[1] != img.size[1]
                       || size[2] != img.size[2])
  {
    // The sizes are not equal, return 0.
    return 0;
  }

  // Compare the image data.
  for(i=0; i < (size[0] * size[1]); i=i+1)
  {
    if(imagedata[i] != img.imagedata[i])
    {
      // There is a difference between the images.
      return 0;
    }
  }

  // If we get here, the images are equal.
  // Note: we don't check to see if the names are equal.
  return 1;

}
```

```
// The purpose of the color_replace function is for
// changing all occurrences of the old color to the new color.
void NPSimage::color_replace(unsigned int oldclr, unsigned int newclr)
{

  int i;  // Loop temp


  // Compare the image data.
  for(i=0; i < (size[0] * size[1]); i=i+1)
  {
    if(imagedata[i] == oldclr)
    {
      // We have the old color, replace it.
      imagedata[i] = newclr;
    }
  }

}
```

```
/*******************************************************************************
* This is TextureDisplayList.C                                                 *
*                                                                              *
*       This program is written to support the texturing need of the software Here we  *
* used the display list phenomena to make the switching of texturing faster. Without  *
* display list it is almost imposible to switch in a short time period though we developed *
* this program under Reality Engines. This Program is also very flexable when more  *
* textures are needed simply call "makeGLTextureFromImagefile("???.rgb") " function  *
* with necessary "rgb" file.                                                   *
*                                                                              *
*                                                                              *
*                                                                              *
*       Authors :       Ltjg. Mustafa YILMAZ                                   *
*                       Ltjg. Mehmet GORGULU                                   *
*                                                                              *
*******************************************************************************/

#include <GL/gl.h>              // Get the OpenGL required includes.
#include <GL/glu.h>
#include <GL/glx.h>

#include "TextureDisplayList.h"
#include "texturesupport_funcs.h"  // Get the texture support functions.
#include "drawsupport_funcs.h"     // Get the drawing functions.
#include "eotda_globals_for_all_programs.h"

void CallDisplayListForCube()
{
  CUBE = glGenLists(1) ;
  glNewList(CUBE,GL_COMPILE) ;
  drawCube(0.0, 0.0, 0.0, 17998.0);
  glEndList();
}



void CallDisplayListForTexturing()
{
         Index = glGenLists(11) ;

        glNewList(Index,GL_COMPILE) ;
                makeGLTextureFromImagefile("sea.rgb");glEndList();

        glNewList(Index + 1,GL_COMPILE) ;
                makeGLTextureFromImagefile("carpet1.rgb");glEndList();


        glNewList(Index + 2,GL_COMPILE) ;
                makeGLTextureFromImagefile("cement4.rgb");glEndList();

        glNewList(Index + 3,GL_COMPILE) ;
                makeGLTextureFromImagefile("sand3.rgb");glEndList();
```

```
    glNewList(Index + 4,GL_COMPILE) ;
            makeGLTextureFromImagefile("brick3.rgb");glEndList();

    glNewList(Index + 5,GL_COMPILE) ;
            makeGLTextureFromImagefile("clouds2.rgb");glEndList();

    glNewList(Index + 6,GL_COMPILE) ;
            makeGLTextureFromImagefile("rocks.rgb");glEndList();

    glNewList(Index + 7,GL_COMPILE) ;
            makeGLTextureFromImagefile("window.rgb");glEndList();

    glNewList(Index + 8,GL_COMPILE) ;
            makeGLTextureFromImagefile("roof.rgb");glEndList();

    glNewList(Index + 9,GL_COMPILE) ;
            makeGLTextureFromImagefile("wood.rgb");glEndList();

    glNewList(Index + 10,GL_COMPILE) ;
            makeGLTextureFromImagefile("city.rgb");glEndList();


}
```

```
/*************************************************************************
* This is "drawsupport_funcs.h " header file.This header file is included only    *
* in "drawsupport.C" file..                                                        *
*************************************************************************/


void drawTarget(int Target_no,float x,float y ,float z);

void drawGround(float x, float y, float z ,int Terrain,int Mode);

void DrawTheSun(int TIME);

void DrawTheColorScale(int MODE);

void DrawTheSyclicColorScale();

void drawSphere(float x, float y,float z,
                float radius, int nslices, int nstacks);

void draw_a_rectangle(int x,int y);

void DrawTheMoutains(int Mode);

void drawCube(float x, float y, float z, float sidelength);

void drawTargetShadow(int Target_no, float x ,float y ,float z);

void update_green_blue(double * RGB_ARRAY , int MODE);
```

```
/**************************************************************************
*       This is file eotda_funcs.h It is the header file for functions defined   *
* in eotda_main.C file.                                                          *
*                                                                                *
*       Authors :       Ltjg. Mustafa YILMAZ                                     *
*                       Ltjg. Mehmet GORGULU                                     *
**************************************************************************/


/**************************************************************************
*                           ---CALBACKS---                                       *
**************************************************************************/

static void rotatexCB(Widget, XtPointer user_data, XtPointer);

static void rotateyCB(Widget, XtPointer user_data, XtPointer);

static void rotatezCB(Widget, XtPointer user_data, XtPointer);

static void translatezCB(Widget, XtPointer user_data, XtPointer);

static void targetCB(Widget, XtPointer user_data, XtPointer);

static void Background1CB(Widget, XtPointer user_data, XtPointer);

static void Background2CB(Widget, XtPointer user_data, XtPointer);

static void Background3CB(Widget, XtPointer user_data, XtPointer);

static void Background4CB(Widget, XtPointer user_data, XtPointer);

static void SensorCB(Widget, XtPointer user_data, XtPointer);

static void initCB (Widget w, XtPointer, XtPointer );

static void exposeCB(Widget w, XtPointer, XtPointer );

static void resizeCB (Widget w, XtPointer, XtPointer call_data );

static void inputCB(Widget, XtPointer, XtPointer call_data );

static void quitCB(Widget w, XtPointer, XtPointer);

static void newviewCB(Widget, XtPointer, XtPointer);

static void DisplayCB(Widget, XtPointer, XtPointer);

static void BackToInputScreenCB(Widget, XtPointer, XtPointer);

static void materialCB(Widget, XtPointer user_data, XtPointer);

static void TargetAreaCB(Widget, XtPointer user_data, XtPointer);
static void TargetHeadingCB(Widget, XtPointer user_data, XtPointer);
static void TargetSpeedCB(Widget, XtPointer user_data, XtPointer);
static void FireStatusCB(Widget, XtPointer user_data, XtPointer);
static void EngineStatusCB(Widget, XtPointer user_data, XtPointer);
static void IsolationStatusCB(Widget, XtPointer user_data, XtPointer);
static void RelativeApertureCB(Widget, XtPointer user_data, XtPointer);
static void ApertureDramCB(Widget, XtPointer user_data, XtPointer);
static void MagnificationCB(Widget, XtPointer user_data, XtPointer);
static void OpticalTransmittanceCB(Widget, XtPointer user_data, XtPointer);
static void ElectronicBandwidthCB(Widget, XtPointer user_data, XtPointer);
static void SensorHeightCB(Widget, XtPointer user_data, XtPointer);
static void DetectivityCB(Widget, XtPointer user_data, XtPointer);
static void DetectorElementsCB(Widget, XtPointer user_data, XtPointer);
static void FiringAngleCB(Widget, XtPointer user_data, XtPointer);
static void FiringDistanceCB(Widget, XtPointer user_data, XtPointer);
static void ChangeWindSpeedCB(Widget, XtPointer user_data, XtPointer);
static void WindDirectionCB(Widget, XtPointer user_data, XtPointer);
static void AeresolCB(Widget, XtPointer user_data, XtPointer);
static void ChangeTimeHourCB(Widget, XtPointer user_data, XtPointer);
static void ChangeTimeMonthCB(Widget, XtPointer user_data, XtPointer);
static void ChangeTimeYearCB(Widget, XtPointer user_data, XtPointer);
static void LongitudeCB(Widget, XtPointer user_data, XtPointer);
static void LatitudeCB(Widget, XtPointer user_data, XtPointer);
static void CloudinessCB(Widget, XtPointer user_data, XtPointer);
static void FogCB(Widget, XtPointer , XtPointer);
static void FogDensityCB(Widget, XtPointer , XtPointer);
static void IR_ScaleCB(Widget, XtPointer , XtPointer);
```

```
/**************************************************************************
*                              ----FUNCTIONS---                           *
**************************************************************************/

GLboolean drawWP();

void draw_the_scene();

void Create_The_Scales(Widget Parent) ;

void BuildInputScreenDialog();

void create_target_screen_widgets (Widget Parent);

void create_sensor_screen_widgets (Widget Parent);

void create_background_screen_widgets (Widget Parent);

void create_others_screen_widgets (Widget Parent);

void create_bulletin_boards(Widget) ;

void concatenate_accumulative_matrices(float refx, float refy, float refz);

void unit(GLfloat *m);

void set_initial_viewing_values();

void compute_new_viewing_values();

void make_the_window_for_the_controls();

void create_toggle_widget(Widget parent);

void load_the_viewpoint();

int  FillInInitialTargetArrays(char * InputFileName);

void make_pulldown_entry(
   Widget menupane,                    // A menupane of the menubar.
   char *entrytext,            // Display text for the pulldown entry.
   XtCallbackProc callback,            // Name of procedure to call
                                       // when this menu entry is selected.
   XtPointer user_data);      // Data to be sent the callback.


 void Read_Screen();

 void Read_TV_Screen();

 float Calculate_Extinction();

 int Fill_Up_Color_Table(int Vertice_no,int Color_Mode);
```

```
void Modify_Color_Array_With_A_Given_Mode(int Vertice_no,int SELECTED_MODE, double
red_value);

 float Find_Total_Radiance(float kt);

 float Temp_At_Sensor(float total_radiance , float mu );

int LoadModifiedGroundColor(float Range, float mu);

int Find_Index_For_Max_Value();

void Update_ROW_COLOMN(int, int);

void Print_Lock_On(int MAX_X, int MAX_Y);

void Call_IR_Functions();

void Call_TV_Functions();

void Print_Numbers_On_Color_Scale();
```

```
/*************************************************************************
*                                                                        *
*       This is  eotda_mains.h file.Here all necessary global widgets and constants are  *
* defined which are used in eotda_main.C                                 *
*                                                                        *
*       Authors :         Ltjg. Mustafa YILMAZ                           *
*                         Ltjg. Mehmet GORGULU                           *
*                                                                        *
*************************************************************************/
#ifndef __EOTDA_GLOBALS__
#define __EOTDA_GLOBALS__


/*************************************************************************
*                 Global widgets definitions                             *
*************************************************************************/


Widget
        toplevel,               // The top level (shell) widget.
        MainWindow ,            // The topmost container widget (excluding the shell).
                                // This type of widget is used as a container for a
                                // collection of widgets that dynamically resize
                                // together.
        InputWindow = 0,        // This is the main widget that we use for input
                                // window.
        DisplayWindow = 0,      // This is the main widget that we use for display
                                // window.
        ControlWindow ,
        DisplayButton ,
        bulletin_board1,
        bulletin_board2,
        bulletin_board3,
        bulletin_board4,
        BboardForExit ,
        BboardForReset,         // Bulleten boards for reset and exit buttons.
        BboardForInput,         // B.B. for InputScreen button..
        MainDisplayFrame,               // Freame that holds the DisplayWindow
        frame,                          // Frame widget attached to Form widget.
        frame1,
        frame2,
        frame3,
        frame4,                 // Frame widget attached to bulletin_board
                                //widgets.
        glw,            // The GL widget that we use to draw into.
        info ,
        scale1,
        scale2,
        scale3,
        scale4,         // Scale widget.
        controls,               // Form widget for the toggle controls.
        rc ,                    // The bulleten board widget for control window.
        rc2 ,                   // The bulleten board for pushbuttons in display
                                // screen
        sep,                    // The seperator widget.
        SensorRC11,
        TargetRowColumn3,
        TargetRowColumn4,
        TargetBB5,
        TargetBB6,
        TargetBB7,
        FogBB ,
        WindSpeedLabel,                  // The label Gadget holds the wind speed.
        TimeLabelHour ,          // The label Gadget holds the Hour
        TimeLabelMonth,                  // The label Gadget holds the Month.
        TimeLabelYear ,          // The label Gadget holds the Year.
        FiringAngleTextWidget,
        FiringDistanceTextWidget;


/************************************************************************
*                   WINDOW VARIABLES                                    *
************************************************************************/

static XtAppContext app_context;      // Applications context.

static XtWorkProcId workprocid = NULL;  // Id of the work procedure

static GLXContext glx_context;        // A GL context (see initCB).

Display *global_display;             // Global display.

Window global_window;                 // Global window.

static XmStringCharSet charset = (XmStringCharSet) XmSTRING_DEFAULT_CHARSET;

/************************************************************************
*                   CONSTANTS                                           *
************************************************************************/
#define sigma 5.6697*pow(1.0,-12.0)
#define PI 3.1415926545
#define Speed_Of_Light  3*pow(1.0,8.0)
#define Boltzman_Constant 1.38*pow(1.0,-23.0)
#define hc 1.986*pow(1.0,-25.0)
#define Integral_Constant 1.1916*pow(1.0,-16.0)
#define PT 100
#define STROKE 101
#define END 102

/************************************************************************
*                   CLASS DEFINITIONS                                   *
************************************************************************/

class    winddata {
         public:
                  int      value      ;
                  int      direction  ;
                   };
```

```
class    timedata {
         public:
                  int      hour       ;
                  int      month  ;
                  int      year   ;
                    };

typedef struct charpoint {
   GLfloat   x, y;
   int    type;
}CP ;



/*************************************************************************
*                         VARIABLES                                      *
*************************************************************************/


          int           n , i      ;         // Used to specify number of args.
static float            tz         ;         // translation on in the z direction.
static float            pitch      ;         // rotation on in the x direction.
static float            heading;             // rotation on in the y direction.
static float            roll       ;         // rotation on in the z direction.

static float            density ;            // used for fog density.
static float            R_values[27][4] = {0.0};  // This array is keeping the red color values
                                                     // for each box which is 100 by 100 pixel.
float    MAX_RED = 0.0,
         Max_Contrast_Of_Scene = 0.0;
int      MAX_X = 0,
         MAX_Y = 0,
         Max_Contrast_X,
         Max_Contrast_Y,
         ROW = 0,
         COLOMN = 900 ,
         OLD_MAX_X = 1,
         OLD_MAX_Y = 1,
         INDEX_FOR_MAX_RED,
         READ_ONES = 0,
         DISPLAY_ENABLED = 0,          // We use it to control displaye screen functions.
         SEARCHED_ONES = 0,
         BackGround = 35,       // Globally set background foR default value which is SOIL
         Target = 29,                  // Set default target value which is TANK
         HOUR = 0 ,                    // Used in positioning the sun function.
         MODE = 202 ;    // Default scale is temperature scale.
 GLint FOG_FLAG ;
```

```
/**************************************************************************
*                         VARIABLES FOR INPUT SCREEN                      *
**************************************************************************/

static float    Firing_Distance = 400.0,// Default distance
                Firing_Angle = 10.0 ,     // Apply a default value (in degrees)
                                          // measured from +y axis towards x plane.
                Target_Area,
                Target_Heading = 0,       //Heading from North in degrees
                Target_Speed,
                Relative_Aperture,
                Aperture_Dram,
                Magnification,
                Optical_Transmittance,
                Eln_Band_Width,
                Sensor_Height,
                Detectivity,
                Wind_Speed,
                Wind_Direction,
                Latitude,                 //Lat and Long. values would better be
                Longitude;                //defined in a class so degrees and
                                          // minutes values could be stored seperately.


/**************************************************************************
*                 VARIABLES FOR TARGETS                                   *
**************************************************************************/

// Globals for targets status.
        int     ENGINE_ON = 1,                  // For Tank Targets.
                FIRED    = 1,                   // diddo.
                HEAT_ISOLATED = 1 ,  // For buildings.
                PARKED = 1 ;         // For Aircraft .


// These variables are used to control loop control variables upper limit, which
// is different for different targets.
        int     Tank_Index = 1384       ,
                Airplane_Index = 208    ,
                Ship_Index = 629,
                Building_Index = 38     ,
        VERTICE_NUM  = Tank_Index ; //Default value for num_of_vertices.


#endif
```

```
/***********************************************************************
*                                                                      *
*        This is  eotda_globals.h file.Here all necessary global variables and constants are *
*  defined which could be refered in all .C files                      *
*                                                                      *
*        Authors :          Ltjg. Mustafa YILMAZ                       *
*                           Ltjg. Mehmet GORGULU                       *
*                                                                      *
***********************************************************************/
#ifndef __COMMON__
#define __COMMON__ extern
#endif

__COMMON__
        float    Vertex[2500][3],          // Array to hold the coordinates.
                 Color [2500][4],          // Array to hold RGBAlpha values.
                 Normal[2500][3],                  // Array to hold Normal array values.
                 Temperature  [2500][1],   // Array to hold Temperature for each veretex.
                 Original_Temp[2500][1],   // Array to hold initial temp.
                 Viewpoint[3] ,            //-x-y-z coordinates for viewer.
                  Refpoint [3] ,           //x-y-z coordinates for the point looked towards.
                  avarage_red_datas[110][3] ,
                 GroundColor[4]  ;

__COMMON__
GLuint  Index,
                 BACKGROUND ,
                 MOUNTAIN  ,
                 CLOUDS    ,
                 CUBE      ;

__COMMON__  int      SENSOR = 39;              // Default sensor is IR.

#define  IR 39
#define  TV 40
#define  LASER 41
#define  GRAY_SCALE 200
#define  SYCLIC_SCALE 201
#define  TEMPERATURE_SCALE 202

static int _IR = IR ;
static int _TV = TV ;
static int _LASER = LASER ;
static int _GRAY_SCALE = GRAY_SCALE ;
static int _SYCLIC_SCALE = SYCLIC_SCALE ;
static int _TEMPERATURE_SCALE = TEMPERATURE_SCALE ;
```

```
/************************************************************************
* This is "global_targets.h " header file. Here are the global definitions for targets and  *
* environment textures.                                                                   *
************************************************************************/
#ifndef _GLOBAL_TARGETS_H
# define _GLOBAL_TARGETS_H
#define TANK 29
#define TRUCK 30
#define SHIP 100
#define HELO 32
#define HOUSE 33
#define PLANE 34
#define  SOIL 35
#define   GRASS 36
#define   SEA 37
#define  ASPHALT 38



static int _TANK = TANK;
static int _TRUCK = TRUCK;
static int _SHIP = SHIP;
static int _HELO = HELO;
static int _HOUSE = HOUSE;
static int _PLANE = PLANE;
static int _SOIL = SOIL ;
static int _GRASS = GRASS ;
static int _SEA = SEA ;
static int _ASPHALT = ASPHALT ;

#endif
```

```
/****************************************************************************
 *  Below is the original image file definitions written by Paul Haeberli.     *
 ****************************************************************************/
#ifndef  __GL_IMAGE_H__
#define  __GL_IMAGE_H__
#ifdef __cplusplus
extern "C" {
#endif


/*
 *        Defines for image files . . . .
 *
 *                      Paul Haeberli - 1984
 *     Look in /usr/people/4Dgifts/iristools/imgtools for example code!
 *
 */

#include <stdio.h>

#define IMAGIC            0732

/* colormap of images */
#define CM_NORMAL              0        /* file contains rows of values which
                                         * are either RGB values (zsize == 3)
                                         * or greyramp values (zsize == 1) */
#define CM_DITHERED            1
#define CM_SCREEN              2        /* file contains data which is a screen
                                         * image; getrow returns buffer which
                                         * can be displayed directly with
                                         * writepixels */
#define CM_COLORMAP                     3       /* a colormap file */

#define TYPEMASK               0xff00
#define BPPMASK                         0x00ff
#define ITYPE_VERBATIM                  0x0000
#define ITYPE_RLE              0x0100
#define ISRLE(type)            (((type) & 0xff00) == ITYPE_RLE)
#define ISVERBATIM(type)       (((type) & 0xff00) == ITYPE_VERBATIM)
#define BPP(type)              ((type) & BPPMASK)
#define RLE(bpp)               (ITYPE_RLE | (bpp))
#define VERBATIM(bpp)          (ITYPE_VERBATIM | (bpp))
#define  IBUFSIZE(pixels)      ((pixels+(pixels>>6))<<2)
#define  RLE_NOP                        0x00

#define  ierror(p)        (((p)->flags&_IOERR)!=0)
#define  ifileno(p)            ((p)->file)
#define  getpix(p)             (--(p)->cnt>=0 ? *(p)->ptr++ : ifilbuf(p))
#define putpix(p,x)            (--(p)->cnt>=0 \
                                  ? ((int)(*(p)->ptr++=(unsigned)(x))) \
                                  : iflsbuf(p,(unsigned)(x)))

typedef struct {
    unsigned short imagic;        /* stuff saved on disk . . */
    unsigned short          type;
    unsigned short          dim;
    unsigned short          xsize;
    unsigned short          ysize;
    unsigned short          zsize;
    unsigned long   min;
    unsigned long   max;
    unsigned long   wastebytes;
    char            name[80];
    unsigned long   colormap;

    long            file;               /* stuff used in core only */
    unsigned short          flags;
    short           dorev;
    short           x;
    short           y;
    short           z;
    short           cnt;
    unsigned short *ptr;
    unsigned short *base;
    unsigned short *tmpbuf;
    unsigned long   offset;
    unsigned long   rleend;             /* for rle images */
    unsigned long   *rowstart;          /* for rle images */
    long            *rowsize;           /* for rle images */
} IMAGE;

IMAGE *icreate();
/*
 * IMAGE *iopen(char *file, char *mode, unsigned int type, unsigned int dim,
 *              unsigned int xsize, unsigned int ysize, unsigned int zsize);
 * IMAGE *fiopen(int f, char *mode, unsigned int type, unsigned int dim,
 *              unsigned int xsize, unsigned int ysize, unsigned int zsize);
 *
 * ...while iopen and fiopen can take an extended set of parameters, the
 * last five are optional, so a more correct prototype would be:
 *
 * IMAGE *iopen(char *file, char *mode, ...);
 * IMAGE *fiopen(int f, char *mode, ...);
 *
 * unsigned short *ibufalloc(IMAGE *image);
 * int ifilbuf(IMAGE *image);
 * int iflush(IMAGE *image);
 * unsigned int iflsbuf(IMAGE *image, unsigned int c);
 * void isetname(IMAGE *image, char *name);
 * void isetcolormap(IMAGE *image, int colormap);
 * int iclose(IMAGE *image);
 *
 * int putrow(IMAGE *image, unsigned short *buffer, unsigned int y, unsigned int z);
 * int getrow(IMAGE *image, unsigned short *buffer, unsigned int y, unsigned int z);
 *
 */

IMAGE *iopen(const char *file, char *mode, ...);
IMAGE *icreate();
unsigned short *ibufalloc();

#define IMAGEDEF                /* for backwards compatibility */
#ifdef __cplusplus
}
#endif
#endif  /* !__GL_IMAGE_H__ */
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center		2
Cameron Station		
Alexandria, VA	22304-6145	
2. Library, Code 52		2
Naval Postgraduate School		
Monterey, CA	93943-5002	
3. Prof. W. R. Colson, Code PH/CW		1
Department of Physics		
Naval Postgraduate School		
Monterey, CA	93943-5100	
4. Prof. Richard Chris Olsen		3
Code PH/05		
Department of Physics		
Naval Postgraduate School	93943-5100	
5. Prof. David R. Pratt		1
Code CS/PR		
Computer Science		
Naval Postgradute School	93943-5100	
6. Deniz Kuvvetleri Komutanligi		1
(Turkish Naval Headquarters)		
Personel Daire Baskanligi		
Bakanliklar, Ankara / TURKEY		
7. Golcuk Tersanesi Komutanligi		2
(Golcuk Shipyard)		
Golcuk, Kocaeli / TURKEY		
8. Deniz Harp Okulu Komutanligi		2
(Naval Academy)		
81704 Tuzla, ISTANBUL / TURKEY		